Apprentissage du style musical et interaction sur deux échelles temporelles

Nicolas DURAND
DEA ATIAM
Université Pierre et Marie Curie, PARIS VI

Laboratoire d'accueil : IRCAM, Équipe Représentations Musicales
Responsable : Gérard Assayag

30 juin 2003

TABLE DES MATIÈRES

I. INTRODUCTION 7

II. ÉTAT DE L'ART 9

II.1. MODÉLISATION DU STYLE MUSICAL 9
II.2. LE « CONTINUATEUR » 11
II.3. GENJAM 11

III. MODÈLE GÉNÉRAL 13

III.1. COMPRESSION À BASE DE DICTIONNAIRES 14
III.1.1 ALGORITHME À FENÊTRE COULISSANTE : LZ77 14
III.1.2 ARBRE DE SUFFIXES. 15
III.1.3 LZ78 17
III.1.4 PST : PROBABILISTIC SUFFIX TREE 18
III.2. ORACLE DES FACTEURS 20
III.2.1 LES AUTOMATES 20
Définition 20
III.2.2 L'ORACLE DES FACTEURS 22
Les facteurs 22
Construction de l'oracle des facteurs. 22
III.2.3 ALGORITHME SÉQUENTIEL 23
Vecteur de suppléances, liens suffixes 24
III.2.4 ORACLE ET ARBRE DES SUFFIXES 25
III.2.5 VECTEUR « LENGTH REPEATED SUFFIX » (LRS) 34
Construction du vecteur lrs 34
III.2.6 MODALITÉS DE NAVIGATION 35

IV. ARCHITECTURE DU SYSTÈME OMAX 37

V. MAX/MSP ET MIDISHARE 39

VI. L'ORACLE DES FACTEURS DANS OPEN MUSIC 43

VI.1. LA CLASSE ORACLE 43
VI.2. FONCTIONS DE BASE DE LA CLASSE ORACLE 44
VI.3. REPRÉSENTATIONS INFORMATIQUES 44
VI.3.1 MODE FREE 44
Champs de la classe « CrossEvent » 46
VI.3.2 MODE BEAT 46
Grammaire de substitution des accords 47
Champs de la classe « Beat » 48

VII. VERS UN ORACLE INVENTIF 49

VII.1. CONSIDÉRATIONS TEMPORELLES 49

VII.2. UN ORACLE « HYBRIDE » 50
VII.2.1 L'ORACLE RYTHMIQUE 50
Les rytbeats 51
VII.2.2 ARBRE RYTHMIQUES 52
VII.2.3 LABELS HARMONIQUES RELATIFS 53
VII.2.4 DES ORACLES MÉLODIQUES : « L'ORACK » 54
VII.2.5 RECOLLEMENT RYTHMICO-MÉLODIQUE 55
VII.2.6 TESTS 56

VIII. PROSPECTIVES 57

Utilisation de l'Oracle 57
Oracles parallèles 57
Protocole de communication MAX / Open Music 58
Recombinaisons rythmico-mélodiques 58
Mode free-beat 58

IX. APPENDICE (CODE LISP) 61

Définitions des classes d'objets 61

X. BIBLIOGRAPHIE 63

Je tiens à remercier sincèrement l’ensemble de l’équipe Représentations Musicales de l’IRCAM pour son accueil.

Et plus particulièrement,

Merci à Gérard Assayag et à Marc Chemillier pour leur disponibilité, leurs encouragements et pour la latitude qui m’a été laissée dans le choix de l’orientation de mon travail.

Merci à Georges Bloch et Karim Haddad pour les discussions et les échanges qui ont apporté de nouveaux éclairages à mon travail.

Merci à Olivier Lartillot, Benoît Meudic et Jose Diago que j’ai côtoyés quotidiennement, pour leur support technique et moral.

Merci enfin à l’ensemble de l’équipe doctorale ATIAM et aux élèves en compagnie desquels j’ai passé cette année.

I. INTRODUCTION

Les développements connus par l'informatique musicale au cours des 20 dernières années, facilités par l'envolée des puissances de calcul disponibles, mettent aujourd'hui la machine au premier plan de la création artistique. L'informatique prend désormais tout naturellement part à la scène contemporaine des musiques écrites, populaires, improvisées.

Un des objectifs de l'IRCAM (Institut de Recherche et de Coordination Acoustique et Musique) est d'apporter aux musiciens et compositeurs d'aujourd'hui les outils et matériaux nécessaires à la prospection de ces nouveaux domaines. L'équipe « Représentations Musicales » de l'IRCAM dirigée par Gérard Assayag met sur pied de tels outils, destinés au traitement informatique de structures musicales de haut niveau (on pense en particulier au logiciel Open Music), principalement à l'usage des compositeurs.

De plus, de tels travaux servent aussi à doter la machine d'une capacité spécifique de traitement de l'information musicale. C'est dans ce créneau particulier que s'insère l'objet du présent rapport.

G. Assayag, M. Chemillier, G. Bloch, O. Lartillot, et E. Poirson entre autres ont, depuis 2001, mené ou participé au cheminement d'un ambitieux projet d'interaction homme-machine. Le paradigme que celui-ci explore est fondé sur des méthodes « universelles » d'apprentissage non supervisé (IP, PST, LZ, et désormais l'Oracle des facteurs). La problématique fondamentale en est la caractérisation du style musical dans un cadre aussi mouvant que celui de l'improvisation, et son appropriation par la machine. Peut-on, par des méthodes appropriées d'organisation du matériau musical, simuler des improvisations « dans le style » d'un improvisateur humain?

Les travaux menés dans ce cadre, jusqu'au stage de E. Poirson concernent la simulation du style musical *en temps différé et sans contrôle harmonique ni rythmique*. Les expériences réalisées avaient pour but de valider un modèle d'apprentissage non supervisé, avec un parti-pris agnostique. Autrement dit, il s'agissait de vérifier la pertinence d'un système sans connaissances musicales, capable de "découvrir" automatiquement des structures musicales par un mécanisme d'apprentissage « universel ».

A partir de 2003, G. Assayag et M. Chemillier ont construit sur ces bases une architecture temps-réel permettant une véritable interaction homme-machine. Le système baptisé OMAX met en conjonction MAX/MSP et Open Music au sein d'un protocole d'échange de données particulier. L'idée directrice est de réaliser un apprentissage en temps-réel à partir des improvisations d'un musicien humain, et

de pouvoir le faire dialoguer avec son « double » simulé. Le système fonctionne en mode totalement libre (instrumentiste solo), ou avec accompagnement (pour des improvisations de type jazz). Ce dernier mode prend en compte la pulsation, et dispose d'un système de substitutions d'accords qui enrichit l'accompagnement.

Les traitements effectués par Open Music sur le matériau musical ont, ces derniers mois, continué à évoluer, bien que l'architecture du système n'ait pas subi de changement majeur. En particulier, on propose un nouveau modèle de recombinaisons rythmico-mélodiques destiné à rendre Omax plus « inventif ». Le travail présenté ici se veut une description approfondie du système Omax de ses débuts à aujourd'hui. On tentera aussi d'apporter un éclairage pertinent sur le sens des choix de design informatique et algorithmique.

Les problématiques majeures abordées dans ce rapport sont :

_ *Les échelles temporelles d'analyse et de restitution*, pierre angulaire du système, puisqu'on se place en même temps dans le cadre de l'interaction *en temps réel* et dans celui de la génération d'improvisation, qui tient plus du *temps différé ou compositionnel*.

_ *Les processus d'innovation* que l'on peut mettre en place pour étoffer les simulations d'improvisation, sans pour autant perdre les éléments déterminants du style musical de l'improvisateur humain.

II. ÉTAT DE L'ART

II.1. Modélisation du style musical

Débutons en citant Lartillot, Assayag, Dubnov et Bejerano [5], à propos de la modélisation du style musical. Ces considérations concernent les premières expériences qu'ils ont mené à l'aide d'algorithmes tels que IP (Incremental Parsing, cf. III.1.3 LZ78) et PST (Probabilistic Suffix Tree, cf. III.1.4 PST).

« Par « Modélisation du Style », nous entendons la construction d'une représentation computationnelle du matériau musical, à même de capturer des traits stylistiques cachés dans la manière dont les patterns rythmiques, mélodiques, harmoniques et polyphoniques sont entremêlés et recombinés de façon redondante. Un tel modèle rend possible la génération de nouvelles séquences musicales respectant le style ainsi explicité (Assayag & al 1999).

Nous sommes donc dans un schéma analyse-synthèse, où la similarité du synthétique à l'original peut être évalué, et ainsi valider le protocole d'analyse. Notre approche est non supervisée, c'est-à-dire que nous voulons un processus d'analyse automatique pouvant fonctionner avec une très grande quantité de données musicales. Parmi les applications intéressantes, on compte les outils de caractérisation du style pour les musicologues (Dubnov & al 1998), la génération de méta-données stylistiques pour la recherche intelligente dans les bases de données musicales, la génération de musique pour des applications au web et aux jeux, l'improvisation machine avec des musiciens humains, et la composition assistée par ordinateur. »

C'est dans cette optique que s'inscrit le développement du système Omax, objet de ce rapport. O. Lartillot souligne dans [3] la proximité entre cette démarche et celle de Nicolas RUWET (cf. [14]), qui cherche à effectuer une analyse musicale par l'utilisation d'un algorithme explicite, et ce de manière *agnostique*. Son approche procède de la linguistique. A propos de sa démarche, il écrit :

« […] Je traiterai la musique comme un système sémiotique, partageant un certain nombre de traits communs – tels que l'existence d'une syntaxe – avec le langage et d'autres systèmes de signes. Je laisserai complètement de côté l'aspect proprement esthétique […].

Dans le premier cas [du message au code], la démarche est analytique ; elle s'impose en principe chaque fois que, s'agissant d'une langue inconnue, […] le message est seul donné.Le travail de l'analyste consiste alors à décomposer et manipuler le corpus (l'ensemble donné de messages) de diverses manières, de façon à dégager les unités, classes d'unités et règles de leurs combinaisons, qui

constituent le code. Le problème crucial est ici celui des procédures de découvertes, c'est-à-dire des critères d'analyse.

[…]

Une fois le code dégagé, une démarche inverse permet de générer des messages à partir de ce code, selon des règles de dérivation qui peuvent, elles aussi, être explicitées rigoureusement. […] Si le modèle analytique est bon, sa transformation synthétique engendrera des messages qui ne figuraient pas dans le corpus initial (limité par définition) mais que les sujets reconnaîtront comme également bien formés. »

On conçoit que cette approche agnostique, fondée sur l'inférence de motifs redondants, soit valide. En effet, en particulier dans le cas de l'improvisation jazz, qui motive notre démarche, les patterns et riffs sont d'un usage traditionnel.

À ce propos, R. Rowe présente dans [8] l'induction de motifs comme une technique générale, dont on peut développer une théorie hors du champ de la musique. Il présente le point de vue de D. Cope dans l'extrait suivant :

« L'induction de motif est l'affirmation que certaines séquences (de nombres) ont été répétées à tel point qu'elles doivent être indiquées comme significatives et conservées, en vue d'une utilisation ultérieure. Les *Experiments in Music Intelligence* (EMI) de David Cope sont des exemples d'induction de motif : de larges échantillons de musique provenant de compositeurs précis sont analysés par le programme pour en extraire des motifs ou des « signatures », qui dépassent les seuils de fréquence maintenus par le logiciel. « Cet article admet que :

> 1) L'appariement de motifs est une technique puissante pour découvrir pourquoi le style de tel compositeur sonne de telle manière;
>
> 2) Les motifs jugés comme identiques et qui surviennent dans les divers morceaux d'un compositeur sont représentatifs et constituent ce que l'auteur appelle des signatures ;
>
> 3) La réutilisation de telles signatures dans la musique composée avec des techniques standard d'harmonie et de contrepoint peut conduire à des imitations dans le style du compositeur.
>
> »
>
> (Affronter 1990,128). »

Suivant l'analyse de D. Cope, le processus de synthèse doit s'accompagner d'une injection de connaissances dans le système. Ses représentations ne sont pas entièrement constitutives de la musique qu'il analyse. En effet, il s'intéresse aux « signatures » qu'il peut extraire de la musique écrite (Mozart et Chopin, par exemple, cf. [15]), dont les structures de haut niveau ne peuvent être captées par une simple analyse motivique.

L'objet de notre analyse convient mieux à une approche agnostique. En effet, l'improvisation en jazz procède plus de la combinaison de motifs et de traits musicaux que de la composition. Les harmonies sont fixées par la grille d'accords sous-jacente. Dans notre cas, on analyse un discours musical construit autour de ces harmonies fixes.

II.2. Le « continuateur »

On doit évoquer l'expérimentation du « continuateur » menée par F. Pachet. Ce système met en œuvre des techniques similaires à celles décrites dans ce document. En particulier, on peut dire que son système applique le même protocole qu'Omax en mode free (cf. VI.3.1). Un instrumentiste improvise seul et sans aucune contrainte. Le système « écoute » l'instrumentiste, et réalise un apprentissage en temps réel, à partir duquel il peut générer des improvisations en forme de réponse (on peut parler de « continuation ») à l'instrumentiste. Le protocole est fondé sur l'inférence de motifs redondants par utilisation d'un modèle markovien à taille variable tel que ceux qu'on décrira en III.1. Notre expérience propose, outre un mode similaire à celui-ci, un mode plus contraint, avec un accompagnement à un tempo fixé. Dans ce cadre qui s'inspire de l'improvisation en jazz, on s'intéresse à la corrélation entre les motifs musicaux inférés, et d'autre part le « beat » imposé et les harmonies sous-jacentes.

II.3. GenJam

J. Biles a développé un système d'apprentissage d'improvisation de jazz similaire à notre approche « avec accompagnement ». Son système, nommé « GenJam », utilise des algorithmes inspirés de la génétique pour faire « muter » le matériau musical assimilé. C'est déjà une différence fondamentale avec notre système Omax qui, jusqu'à maintenant, ne modifiait pas le matériau musica, mais se contentait de le recombiner. Une autre grande différence réside dans l'approche « non agnostique » de J. Biles, puisque GenJam possède des connaissances musicales. De plus, l'apprentissage de son système est supervisé, contrairement à celui d'Omax. Après acquisition du matériau (« écoute » d'un instrumentiste et apprentissage par le système), il fait générer des improvisations par son algorithme de mutation, qui sont « notées » par un auditeur humain. Cela détermine l'évolution du système d'un point de vue génétique.

C'est principalement dans le résultat de son expérience que la démarche de J. Biles peut être comparée à la nôtre : il obtient un système capable de continuer à assimiler du matériau en temps-réel, et d'improviser en duo avec un instrumentiste humain, sur le principe des « trading fours » (4 mesures d'improvisation par musicien, chacun à son tour). La problématique des échelles temporelles multiples est présente dans son protocole. En effet, il pose dans le cadre du temps-réel une situation d'apprentissage et de génération de musique qui nécessite de trouver des unités d'analyse et de composition dont les tailles soient appropriées.

Son idée de transformer le matériau musical assimilé par le système est séduisante, pour l'inventivité du résultat et la stimulation que cela peut créer chez un instrumentiste en situation d'interaction. C'est une problématique que l'on abordera dans la partie VII, d'une toute autre manière.

III. MODELE GENERAL

Les algorithmes de classification des données utilisés dans le cadre de notre expérience sont issus d'études sur la compression des données (développements issus de la théorie de l'information, cf. [3]) ou sur l'analyse de séquences du génome. Comme on l'a évoqué dans le chapitre II, on veut repérer des traits caractéristiques d'un style d'improvisation, c'est-à-dire des motifs récurrents et musicalement pertinents.

Tous les algorithmes testés jusqu'ici par l'équipe de recherche ressortent d'un modèle dit "contexte-inférence". Ce modèle suppose de ramener toute séquence à une source Markovienne inconnue dont elle est une réalisation. On en analyse les propriétés statistiques afin d'imiter son comportement.

Dans le cadre de notre expérience, on cherche à organiser le matériau musical de façon à avoir accès efficacement à ses patterns redondants. On veut de cette manière pouvoir être capable d'inférer les différentes continuations d'une séquence donnée. Dans cette situation, on qualifiera la séquence dont on cherche une continuation de *contexte*. Lors de la génération d'une séquence à partir du matériau assimilé par le système, on infère une continuation à partir du contexte. Cette continuation est alors intégrée au contexte, pour continuer la génération.

Les simples chaînes de Markov ont déjà été usées et abusées auparavant. De plus, ce modèle possède une forte rigidité, qui n'est pas particulièrement adaptée à notre utilisation finale. Divers algorithmes dérivant du modèle markovien ont été testés dans le cadre de notre expérience : LZ, Suffix Tree, PST. Ils offrent des caractéristiques variées : taille du contexte irrégulière, mémoire variable... En revanche, ces algorithmes ont tous un caractère « universel ». Autrement dit, ils permettent aussi bien d'organiser des données textuelles que des séquences adn… et dans notre cas des séquences musicales. Le dernier choix s'est porté sur l'Oracle des facteurs, qui à première vue est tout à fait différent et consiste en la construction d'un automate reconnaissant tous les facteurs d'un mot.

Emilie Poirson a mené, lors de son stage de DEA, une étude cataloguant ces divers algorithmes (cf. [4]). On en propose ici une nouvelle revue mettant en exergue leur appartenance à un modèle commun. En effet, on a pu finalement se rendre compte que toutes ces méthodes d'organisation de données peuvent être réunies dans la catégorie générale des arbres de suffixes. Cela peut se montrer par construction pour les algorithmes LZ et PST. Par contre, le cas de l'oracles des facteurs est plus particulier. On conjecture que l'oracle des facteurs s'apparente à

un arbre de suffixes « écrasé », et réduit à un graphe linéaire. Il en résulte une plus grande efficacité de stockage.

Pour développer encore le travail mené par E. Poirson, on propose l'utilisation d'un outil permettant de calculer la longueur de facteurs redondants grâce à l'oracle des facteurs : le vecteur « length repeated suffix ». On verra plus bas l'utilisation qu'on peut en faire pour déterminer des continuations de contexte maximal lors de la génération d'improvisations.

On développera tout d'abord les caractéristiques des différents algorithmes testés, jusqu'à l'actuel oracle des facteurs, avant de détailler les structures informatiques et musicales manipulées dans le cadre de la simulation d'improvisations.

III.1. Compression à base de dictionnaires

Les algorithmes que l'on présente ici procèdent d'une démarche commune, qui est la constitution d'un dictionnaire permettant d'indexer le texte à compresser. Dans notre cas, on s'intéresse particulièrement à ce dictionnaire, et à son utilisation ultérieure en tant que répertoire de mots du texte original.

III.1.1 Algorithme à fenêtre coulissante : LZ77

LZ est un algorithme de compression de données développé par Jacob Ziv et Abraham Lempel.

L'algorithme LZ77 comprime le texte dans le sens de lecture. On fait coulisser une fenêtre associant le dictionnaire à un tampon de lecture, qui doit être comprimé. Si on trouve dans le tampon un mot qui appartient déjà au dictionnaire, alors, on remplace la séquence par un pointeur vers ce mot du dictionnaire (donnant début et taille de la chaîne).

Le dictionnaire est donc augmenté au fur et à mesure de la lecture.

Ainsi, pour un texte écrit, LZ77 permet de ne pas coder les redondances souvent présentes avec les « le », « de », « …sion », « …ment », etc. Cela soulage la quantité d'information et donc compresse les données. Un autre avantage est que ce codage est simple et très économe dans le cas d'un motif répété plusieurs fois. La décompression est simple et rapide, puisqu'il suffit de « suivre » les pointeurs.

Néanmoins, l'implémentation de cet algorithme pose des problèmes de gestion de tableaux à longueur variable et de pointeurs glissants. De plus, à chaque pas de l'algorithme, on compare l'élément du tampon à tous ceux du dictionnaire. Au début, celui-ci possède peu de mots, donc la comparaison s'effectue rapidement. Mais dès que le dictionnaire s'alourdit, le temps de compression augmente avec le nombre de comparaisons à effectuer. Aussi l'algorithme n'est-il pas efficace pour des données particulièrement longues.

On a évoqué au-dessus l'efficacité quant au traitement des répétitions. Dans un texte écrit, la grande quantité de redondances (articles et mots-clef du texte) nous assure une certaine efficacité à l'algorithme. Mais lorsque le support ne contient pas (ou peu) de répétitions, on code, au final, tous les éléments avec trois champs (position, longueur, mot suivant), ce qui finalement peut multiplier la quantité d'information plutôt que la réduire ! Finalement, l'efficacité de LZ77 varie suivant le support sur lequel on l'utilise (cf. [12] pour exemple).

III.1.2 Arbre de suffixes.

Cet arbre « naïf » rend compte de tous les suffixes *du texte* analysé (généralisation à tous objets analysés séquentiellement). Pour construire l'arbre, on considère séquentiellement tous les suffixes du texte. Pour chacun, on parcourt l'arbre en utilisant séquentiellement les lettres qui le composent. Lorsque la transition que l'on veut effectuer n'existe pas, on la crée.

Les feuilles de l'arbre sont les états terminaux rendant compte de tous les suffixes du texte. Si on considère *tous* les états de cet automate terminaux, il permet de reconnaître tous les préfixes de suffixes (i.e. tous les *patterns*, ou *facteurs*) du texte original.

Il est aisé d'utiliser l'arbre de suffixes pour inférer une continuation. En se donnant un *contexte*, sous la forme d'un pattern (i.e. une séquence d'objets du dictionnaire), on parcourt l'arbre jusqu'à un état rendant compte du contexte. On dispose alors de toutes les continuations possibles, sous la forme des branches permettant de continuer la navigation On note qu'en choisissant une continuation parmi celles-ci, on reproduit *un pattern existant*. L'innovation provient de la capacité du système à mélanger les différents patterns ; cela dépend de la taille du contexte pris en compte. On peut justement envisager de le réduire pour élargir le champ des continuations possibles (si le contexte est « abcd », on peut le réduire à « bcd »).

Considérons par exemple l'arbre de suffixes du mot « ABABABABAABB » représenté figure 1. On cherche à inférer une continuation de deux éléments à la séquence « BABA ». Si notre contexte est « BABA » (état marqué « * »), l'arbre de suffixes nous permet d'inférer les continuations « AB » ou « BA ». Si on réduit notre contexte à la séquence « A » (état marqué « ° »), les continuations possibles seront « AB », « BA », ou « BB ».

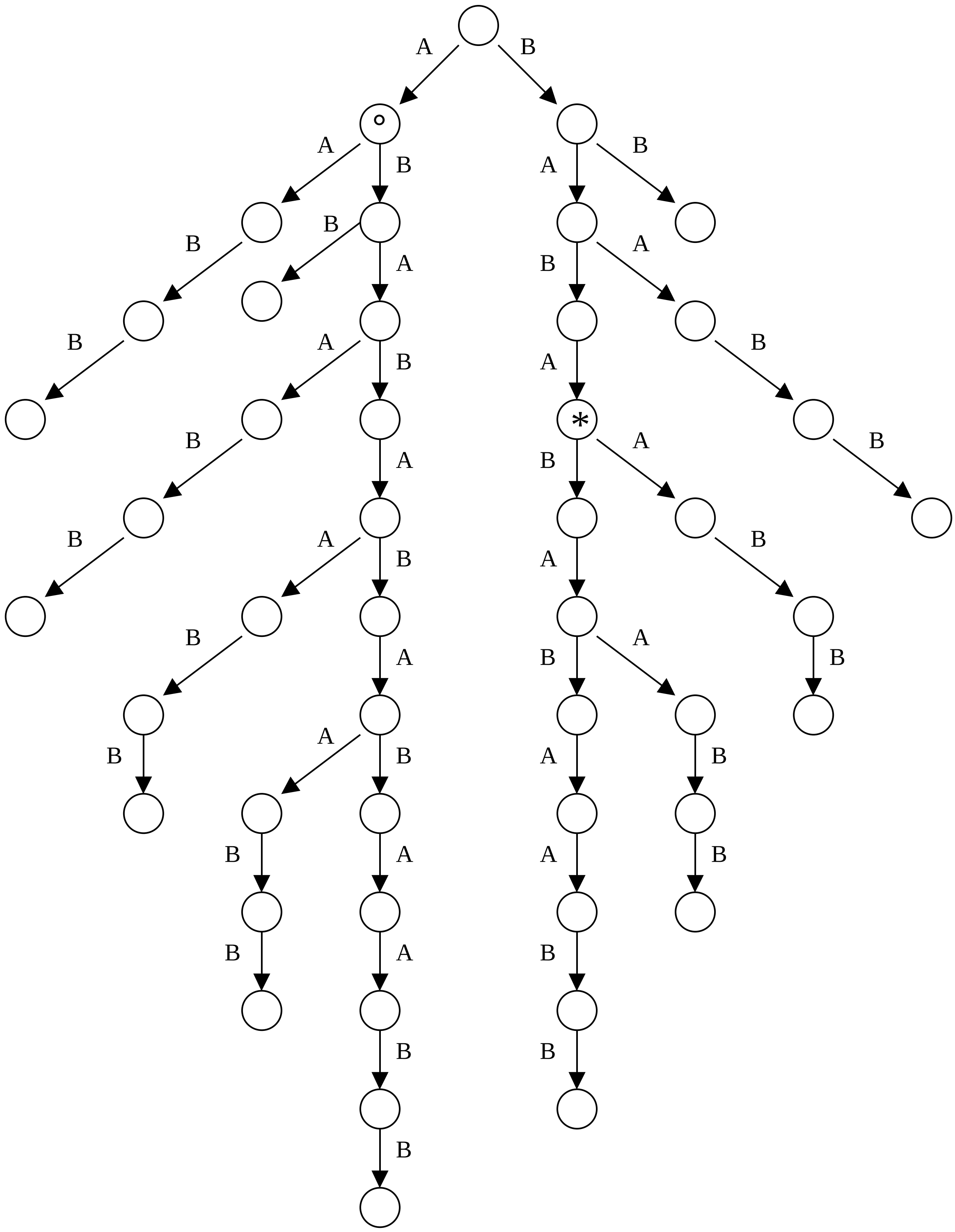


Fig. 1 : Arbre suffixe du mot ABABABABAABB

III.1.3 LZ78

On appelle simplement « LZ » cette version de l'algorithme. C'est une des applications les plus populaires de la théorie de l'information, qu'on retrouve dans des applications de compression de données comme zip ou gzip.

On pourra se référer au rapport de stage de DEA d'O. Lartillot [3], pour des détails sur l' implémentation de LZ dans Open Music.

LZ est initialisé avec un dictionnaire vide. À chaque pas, on cherche la chaîne inconnue la plus longue. Par exemple, pour un texte, l'analyse s'effectue caractère par caractère et sectionne le texte dès qu'une séquence inconnue est trouvée. On commence donc par isoler des séquences très courtes (un seul élément pour les premières). Les séquences que l'on isole sont toujours constituées de :

_ Un suffixe déjà connu (éventuellement vide) référencé par un code (ou existant donc déjà comme une branche de l'arbre que l'on constitue au fur et à mesure).

_ Un préfixe d'un caractère faisant de la séquence totale une séquence inconnue.

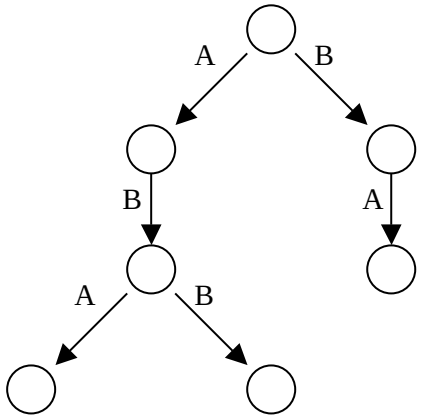
On peut représenter le dictionnaire des séquences sous forme d'un arbre, considéré comme un automate dont tous les états sont finaux.

Chaque séquence que l'on isole reçoit un code, ou vient augmenter d'une branche l'arbre du dictionnaire. On fournit, figure 2, un exemple de dictionnaire constitué par codage LZ d'une séquence de lettres.

Fig. 2 :
Codage LZ du mot « ABABABABAABB »

Le dictionnaire constitué est {A,B,AB,ABA,BA,ABB}

Ci-contre, ce dictionnaire, sous forme d'un arbre.



LZ est un algorithme performant dont les taux de compression ne sont pas négligeables. De plus, il est nettement moins difficile à programmer que LZ77 décrit précédemment. Une autre différence avec son prédécesseur est qu'au lieu de comparer le mot à ceux contenus dans le tampon afin de trouver des répétitions proches, on le compare à tout le dictionnaire. On a ainsi plus de chances de le trouver, et donc de ne pas le coder, ce qui améliore la compression.

Le choix du dictionnaire dépend de la mémoire de la machine. Sur des données courtes, LZ n'est pas particulièrement efficace. En effet, plus le texte à compresser est court, moins l'arbre qui le représente sera développé.

On constate, par construction, que l'arbre-dictionnaire engendré par LZ est inclus dans l'arbre des suffixes. En fait, l'arbre engendré par LZ n'équivaut qu'*asymptotiquement* à l'arbre de suffixes complet. Par rapport à ce dernier, le dictionnaire de LZ est incomplet, puisqu'il ne contient pas les séquences les plus longues. On peut utiliser une Méthode de type « bootstrap » (tirages au sort multiples parmi nos données pour en simuler un plus grand nombre) pour obtenir un comportement « asymptotique ».

On note que, dans notre cas, utiliser un dictionnaire incomplet revient à réduire la taille des contextes pris en compte, ce qui est profitable pour l'innovation dans les séquences que l'on peut générer grâce à cet arbre. On ne trouve donc pas de désavantage particulier, dans le cadre de notre expérience, à utiliser LZ. Au contraire, la quantité de données à stocker est réduite.

III.1.4 PST : Probabilistic Suffix Tree

Le principe est de restreindre le dictionnaire aux éléments apparaissant un nombre significatif de fois dans tout le texte, ce qui implique des pertes d'information.

Il faut ajouter à cela les probabilités. À chaque nœud est associé un vecteur de probabilités décrivant le poids des différentes continuations possibles. Pour construire un PST, il faut donc d'abord connaître l'alphabet. Ensuite, il faut définir L, la longueur de chaîne maximale dans l'arbre. On s'intéresse donc à toutes les sous-chaînes du texte, de longueur maximale L. On fixe également Pmin, seuil de probabilité : si une probabilité est inférieure à Pmin, la chaîne sera considérée comme négligeable. La valeur de Pmin peut varier suivant les utilisations.

Au départ, l'arbre n'est composé que d'un nœud, la racine. À chaque pas, donc pour chaque sous-chaîne, on recherche dans l'arbre si les éléments qui sont susceptibles de « continuer » cette chaîne ont une probabilité supérieure à Pmin, c'est-à-dire s'il existe une continuation non négligeable de la chaîne. On fait de même avec la chaîne privée de son premier caractère. Si les deux conditions sont réunies, on ajoute la chaîne au PST. Sinon, on la considère comme trop peu importante pour s'y intéresser. On fait donc une sélection des mots à garder basée uniquement sur des considérations statistiques. Ceci constitue une des limites de l'algorithme quant à son application à la musique.

Revenons à l'attribution des probabilités. La probabilité que X suive Y est le rapport du nombre de fois où X suit Y au nombre total d'occurrences de Y.

Ex : Pour le mot « ABABABABAABB »,

$$P(B/A) = 5/6$$

(on trouve 5 fois AB pour 6 fois A)
C'est *la probabilité de B sachant A*.

De même,

$$P(B/AA) = 1/1 = 1$$

(on trouve 1 fois AAB pour 1 fois AA)

On a alors *la probabilité de B sachant A ou AA* :

$$P(B/((A)\ \ (AA))) = P(B/A).P(B/AA)$$

Et comme on étudie les chaînes caractère par caractère, on aura, pour le mot « musique » :

$$P(musique) = P(m).P(u/m).P(s/mu).P(i/mus).P(q/musi).P(u/musiq).P(e/musique)$$

La méthode PST est un mode de compression avec pertes, à la différence de LZ étudié auparavant. Cela soulage le dictionnaire mais, comme la sélection se fait sur une fonction de probabilité empirique, on retire le moins fréquent, qui n'est pas nécessairement le moins important. Par exemple, un thème peut être répété plusieurs fois dans un morceau et être joué en miroir une fois à la fin du morceau. L'algorithme nous donnera une faible probabilité du miroir et perdra l'effet écrit par le compositeur. Certaines figures de style qui enrichissent le morceau seront délaissées, ce qui est un problème pour notre travail. Néanmoins, pour la génération d'improvisations, il est particulièrement intéressant d'utiliser des algorithmes non complets, qui peuvent fournir plus d'innovation lors du processus de génération. PST peut être vu comme un arbre de suffixes amputé de ses branches les moins « représentatives ». On a donc là, pour la génération de séquences, un arbre de suffixe réduit, c'est-à-dire un outil imposant des réductions de contexte.

Pour conclure, on peut dire que PST est une méthode plus longue et plus lourde que LZ car on consacre une première étape à calculer, sur l'ensemble du texte, les probabilités d'occurrence, avant de comparer chaque unité une à une dans l'arbre. Cela fait un coût de calcul élevé, et pose des difficultés pour l'implémentation en temps réel. En effet, on veut pouvoir ajouter dans le système le matériau musical nouveau, au fur et à mesure qu'il est joué par l'improvisateur. Cette caractéristique de PST est rédhibitoire pour l'usage que l'on désire en faire : élargir en permanence la base de matériau musical possédée par le système.

III.2. Oracle des facteurs

Dans son acception la plus large, un « texte » peut être, la modélisation aidant, n'importe quelle séquence d'objets pour lesquels on peut définir une fonction de comparaison adéquate. Ainsi, on peut traiter de la même manière la séquence d'acides constituant un brin d'adn, un texte littéraire ou encore une séquence musicale.

Dans ces deux cas, la quantité de données est particulièrement importante, et il est nécessaire de pouvoir les stocker de manière efficace. C'est la raison pour laquelle on trouve des liens entre ce type d'analyse et les algorithmes de compression de données. Les automates semblent, pour le type de stockage des informations qui nous intéresse, constituer un modèle particulièrement efficace d'un point de vue computationnel.

Allauzen, Crochemore et Raffinot donnent dans [1] une version incrémentale (traitement du « texte » à analyser élément par élément) et efficace d'un automate particulier, « l'oracle des facteurs », justement utilisé dans le cadre de la recherche sur le génome. L'oracle des facteurs peut être utilisé afin de repérer les redondances au sein d'un texte, et de reconnaître les facteurs de ce texte.

Emilie Poirson [4] a contribué pendant son stage de DEA à l'implémentation de l'oracle des facteurs dans Open Music, dans le but d'améliorer les performances d'analyse musicale jusque là menées avec l'algorithme LZ. Suivant son approche, on rappellera les principes de base d'automatique avant de développer le fonctionnement de l'oracle des facteurs.

Dans une partie suivante, on expliquera en quoi l'oracle des facteurs peut être apparenté à la classe générale des arbres de suffixes. On y fournit une proposition d'algorithme de transformation de l'arbre de suffixes en oracle.

III.2.1 Les automates

Définition

Un automate fini est défini par un quintuplet {∑, Q, I, F, T} où :

∑ est un alphabet,
Q est un ensemble fini d'états,
I est l'ensemble des états initiaux,
F est l'ensemble des états finaux,
T est un ensemble de règles de transition étiquetées par des éléments de l'alphabet ∑.

Pour analyser un « texte » par le biais de l'automate ainsi défini, on part d'un état initial et on suit séquentiellement les règles de transition s'appliquant pour les éléments du texte.

On a, pour l'exemple figure 3 : :

∑ = {a,b}
Q = {0,1,2,3}
I = {0}
F = {3}
T = {(0, a ,0),(0, b ,1),(1, b ,2),(1, a ,3),(2, (a/b) ,2),(3, (a/b) ,3)}

Les règles de transition se lisent :
(X, ß ,Y) signifie « de l'état X, on passe à l'état Y par une transition étiquetée par ß. ».

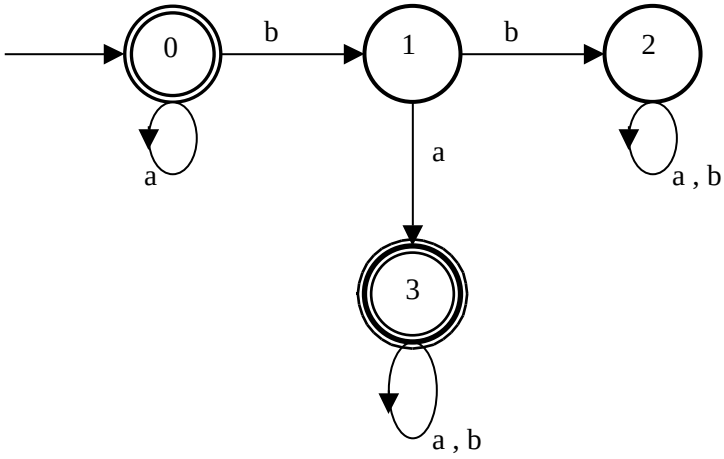


Fig. 3 : Exemple d'automate

Exemples d'analyse de mot par cet automate:
aba : on passe par les états 0-0-1-3. On finit l'analyse du mot en 3, il est donc reconnu par l'automate.
baba : 0-1-3-3-3. Le mot est reconnu.
abba : 0-0-1-2-2. Or (2) n'est pas un état final, le mot n'est pas reconnu par l'automate.

On peut synthétiser la grammaire reconnue par l'automate sous la forme :

G = {a*ba(a/b)*}.

On rappelle que x* est une répétition d'un nombre de fois arbitraire de x. ∑* est l'ensemble des mots que l'on peut constituer avec l'alphabet ∑.

Un automate est dit *déterministe* quand pour tout état, chaque élément de l'alphabet étiquette au plus une transition partant de cet état. Il est dit *complet* quand pour tout état, chaque élément de l'alphabet étiquette exactement une transition partant de cet état. Une grammaire est dite régulière s'il existe un automate fini dont l'ensemble des mots reconnus est exactement celui engendré par la grammaire.
Dans notre exemple, l'automate {∑, Q, I, F, T} est complet et la grammaire G est régulière.

III.2.2 L'oracle des facteurs

Les facteurs

Un mot x ∑* est un facteur du mot p si et seulement si p peut s'écrire p = uxv avec (u,v) $(\sum^*)^2$.

Considérons un mot $p = p_1p_2\ldots p_m$ (suite de lettre prises dans un alphabet ∑). Pour reprendre les termes d'Allauzen, Crochemore et Raffinot [1], « nous cherchons à construire un automate (a) acyclique, (b) qui reconnaisse au moins les facteurs de p, (c) qui possède le moins d'états possibles et (d) qui soit linéaire en nombre de transitions. (…) Nous proposons (…) une construction d'un automate de m+1 états qui satisfait ces quatre critères, que l'on appelle *oracle des facteurs* ».

Les auteurs de cet article conjecturent que le comportement de leur automate est optimal, au vu des conditions exigées. En particulier, l'économie de mémoire est réelle en comparaison avec les arbres de suffixes. De plus, l'algorithme séquentiel proposé est simple d'implémentation.

Construction de l'oracle des facteurs.

On considère un mot de m lettres : $p=p_1p_2\ldots p_m$.

1) On construit m+1 états, numérotés de 0 à m.

2) On crée l'ensemble des transitions $\{(i,p_{i+1},i+1)\}$ pour i [0,m-1].

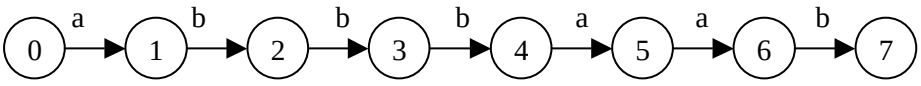


Fig. 4 : Première étape de la construction de l'oracle des facteurs pour le mot « abbbaab »

3) Pour chaque i [0,m-1], on considère un mot u de longueur minimale (cf. [1] pour le lemme garantissant son unicité) reconnu en l'état i. Pour tout σ ∑, $\sigma \neq p_{i+1}$, si uσ est un facteur de p, on construit une transition de p_i vers l'état correspondant à la fin de la première occurrence de uσ.

Tous les états de l'automate ainsi obtenu sont terminaux.

Pour continuer l'exemple précédent, l'oracle des facteurs de « abbbaab » est donné figure 5.

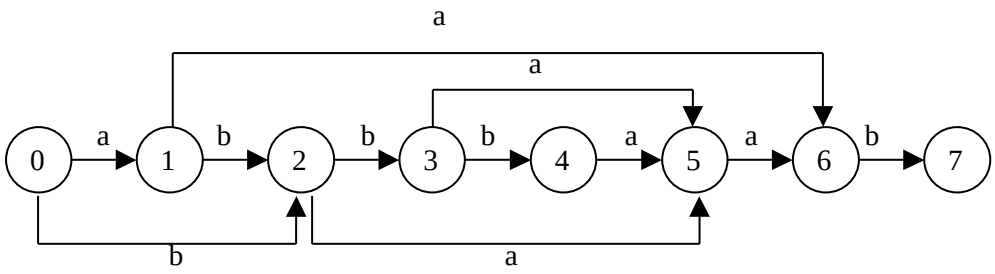


Fig. 5 : Oracle des facteurs du mot « abbbaab »

À titre d'exemple, notons que les mots « aba » et « abaab » sont reconnus par l'oracle des facteurs de abbbaab sans en être des facteurs.

Contrairement aux arbres suffixes qui ne tiennent pas compte des patterns jugés peu significatifs, l'oracle reconnaît tous les facteurs du texte original, et même plus, ce qui, dans notre cas, peut être intéressant puisque cela multiplie les continuations possibles.

III.2.3 Algorithme séquentiel

Allauzen, Crochemore et Raffinot proposent, à la suite de cette pésentation de l'oracle, un algorithme séquentiel pour la construction de l'oracle. Celui-ci permet de construire l'oracle pas à pas, en ajoutant à chaque étape un élément du texte. Autrement dit, on n'a pas besoin de connaître l'intégralité du texte pour construire son oracle. Ce point fait de l'oracle un outil particulièrement adapté à notre protocole d'expérimentation. En effet, on peut ainsi envisager un dialogue entre l'improvisateur et la machine. Cette dernière pourra générer des réponses à faire à l'improvisateur dès l'apprentissage des premières phrases. Cependant, on imagine bien qu'il faut qu'une certaine quantité de matériau ait déjà été assimilée par l'oracle pour que celui-ci développe un vocabulaire musicalement assez riche. L'apprentissage se fera au rythme auquel le musicien fournit ce matériau.

Pour la version séquentielle de la construction entre en jeu la notion de suppléance, qui nous sera utile par la suite. On développera plus bas le sens que l'on peut donner à cet outil de construction. Le vecteur des suppléances, noté Sp[0…x], associe à chaque état de l'automate un pointeur vers un autre état, ou hors de l'automate (-1).

On initialise l'automate avec un état n°0, avec Sp(0) = -1.

À l'étape n, (on note n le nombre de lettres déjà assimilées par l'automate). Pour une lettre σ ajouter, on procède comme suit :

- On crée un nouvel état n°(n+1).
- On crée la transition (n, σ ,n+1).
- La variable k prend la valeur de Sp(n).

· Tant que k est supérieur à –1, et tant qu'il n' y a pas de transition sortant de l'état k étiquetée par σ, alors, on crée la transition (k, σ ,(n+1)) et k reçoit comme nouvelle valeur Sp(k).

Lors de la première itération, k est initialisé à –1, donc on n'entre pas dans cette boucle.

· Pour finir, on attribue à Sp(n+1) la valeur 0 si k = -1 au sortir de la boucle, le numéro de l'arrivée de la transition partant de l'état k et étiquetée par σ sinon.

L'oracle permet de manière très efficace de faire apparaître les redondances présentes au sein d'un « texte » particulièrement long. Cela nous convient tout particulièrement, puisqu'on est amené à traiter une très grande quantité de matériau musical. Le système de suppléances permet de construire la structure d'oracle de manière séquentielle. Autrement dit, c'est grâce à cela que l'on pourra repérer des redondances entre des zones du « texte » aussi éloignées que les premières et les dernières phrases jouées.

Vecteur de suppléances, liens suffixes

Notons que les suppléances constituent en fait une trame de liens parallèles au sein de l'automate, orientés dans le sens des états décroissants.

Pour préciser leur utilité, on se réfère à [1] Allauzen, Crochemore et Raffinot en notant :

$p=p_1p_2\ldots p_m$ le mot analysé.
Pref(p,i) le préfixe $p_1p_2\ldots p_i$ de p constitué de ses i premières lettres.
Repet(p,i) le plus long suffixe de Pref(p,i) qui possède au moins deux occurrences distinctes dans Pref(p,i).

Avec ces notations, on peut définir les suppléances comme suit:

« La fonction de suppléance Sp définie sur les états de l'automate associe à chaque état i>0 l'état j dans lequel se termine la lecture de Repet(p,i). »

Ainsi, les suppléances relient des états en lesquels s'achèvent les reconnaissances de facteurs redondants de taille maximale. Le « chemin suffixe de Pref(p,i)» constitué des états $\{i, Sp(i), Sp^2(i),Sp^3(i),\ldots,0\}$ fournit donc des pointeurs vers de multiples continuations de Pref(p,i). On notera le chemin suffixe de i:

$$SPp(i) = \{i, Sp(i), Sp^2(i),Sp^3(i),\ldots,0\}$$

On a donc avec le vecteur de suppléances un puissant outil de parcours de l'oracle, particulièrement adapté à la génération de séquences nouvelles. On notera que c'est une utilisation détournée d'un vecteur de données utiles pour l'implémentation séquentielle de l'algorithme.

La figure 6 présente de manière simple l'interprétation que l'on peut avoir du vecteur de suppléances. Notons néanmoins que le facteur redondant mis en

évidence sur cette figure *n'est pas* le facteur Repet(p,i) mais un de ses suffixes. L'état Sp(i) n'est en effet que l'état en lequel est *reconnu* Repet(p,i). On exposera plus précisément le sens de cette remarque dans le chapitre suivant concernant la longueur des facteurs redondants pointés par le chemin suffixe.

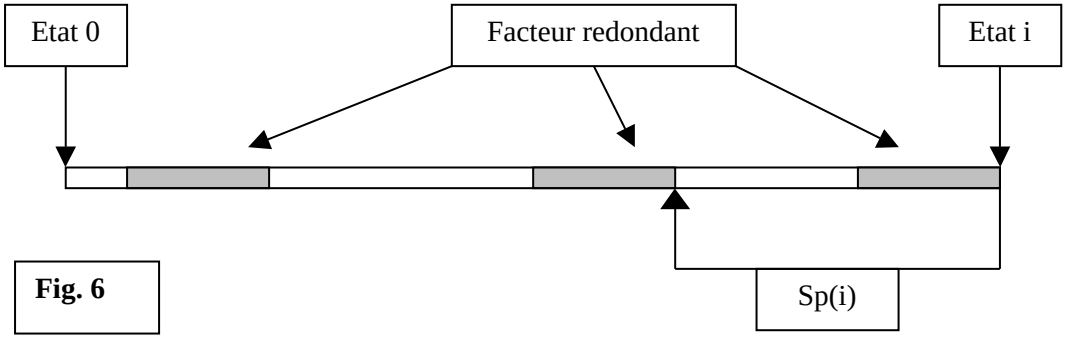


Fig. 6

III.2.4 Oracle et arbre des suffixes

Remarquons que grâce au jeu de liens suffixes mettant en évidence les redondances au sein de la séquence analysée, on perçoit mieux le rapprochement qui peut être fait entre l'oracle (factorisation d'un mot donnant un schéma séquentiel de compression) et les algorithmes de type arbre de suffixes, à l'origine pensés pour la compression des données.

Le jeu de liens suffixes lie des états que l'on peut considérer également représentatifs d'un contexte donné, plus ou moins réduit. En ce sens, on peut voir l'oracle comme un arbre de suffixes réduit à une seule branche. Au lieu d'ajouter des branches lors de la construction, on ajoute des transitions sur la branche principale. L'oracle est un arbre de suffixes particulier, prenant en compte dans sa structure même la réduction du contexte.

On donne figure 7 l'oracle des facteurs du mot « ABABABABAABB ». Pour illustrer la proximité entre l'oracle et l'arbre de suffixes, on propose un algorithme permettant de construire l'oracle des facteurs d'un mot à partir de l'arbre des suffixes de ce mot.

Considérons l'arbre des suffixes du mot $p=p_1p_2\ldots p_m$. On ne conservera in fine que sa plus grande branche, dite maximale. On parcourt cette branche, et pour chaque état j rencontré, on procède comme suit :

- Pour chaque transition sortante p_i, excepté celle qui pointe sur la suite de la branche maximale, on remplace la branche indexée par p_i par la transition (j , p_i , i). On remplace les éventuelles sous-branches de la branche supprimée par leurs équivalentes sur la branche maximale.

L'idée est que chaque branche de l'arbre de suffixes correspond à la reconnaissance d'un suffixe de p. Or la branche maximale contient tous les suffixes. Au lieu d'utiliser des branches annexes pour reconnaître les suffixes, on les redirige donc toutes vers le suffixe auquel elles correspondent respectivement, sur la branche maximale.

On donne l'exemple, figures 8 à 14, de la transformation de l'arbre de suffixes de « ABABABABAABB » en l'oracle des facteurs du même mot.

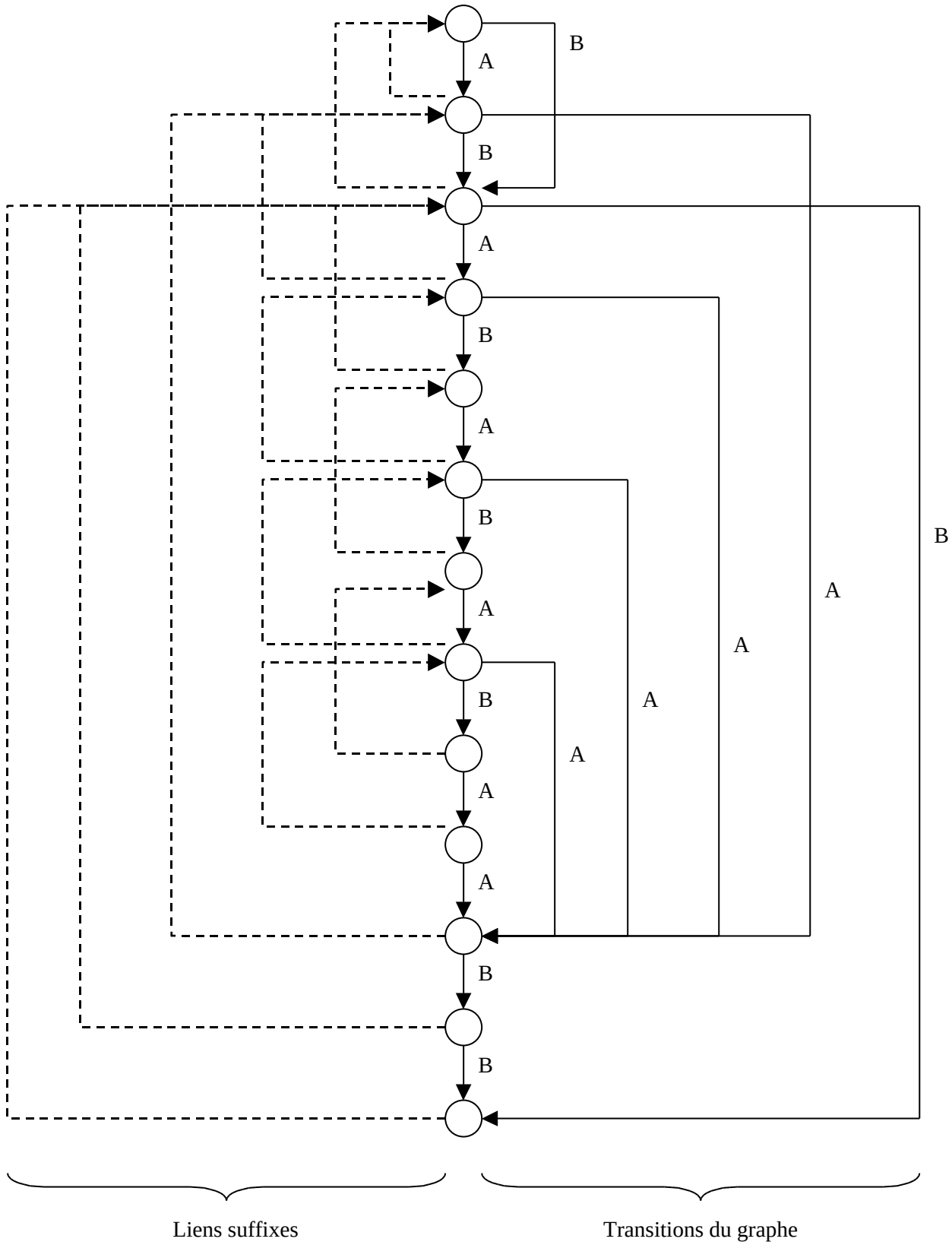


Fig. 7 : Oracle des facteurs du mot ABABABABAABB

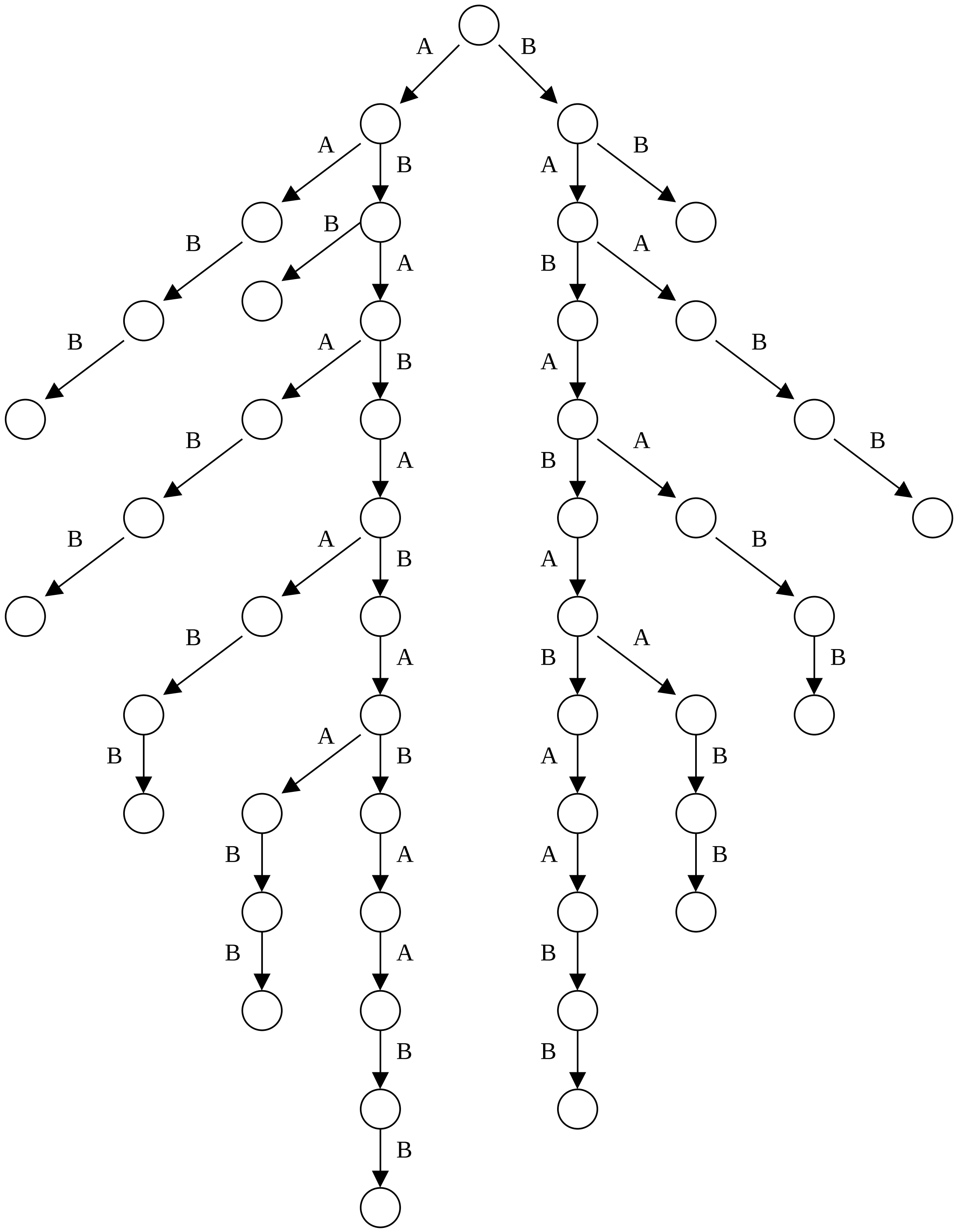


Fig. 8 : Arbre suffixe du mot ABABABABAABB

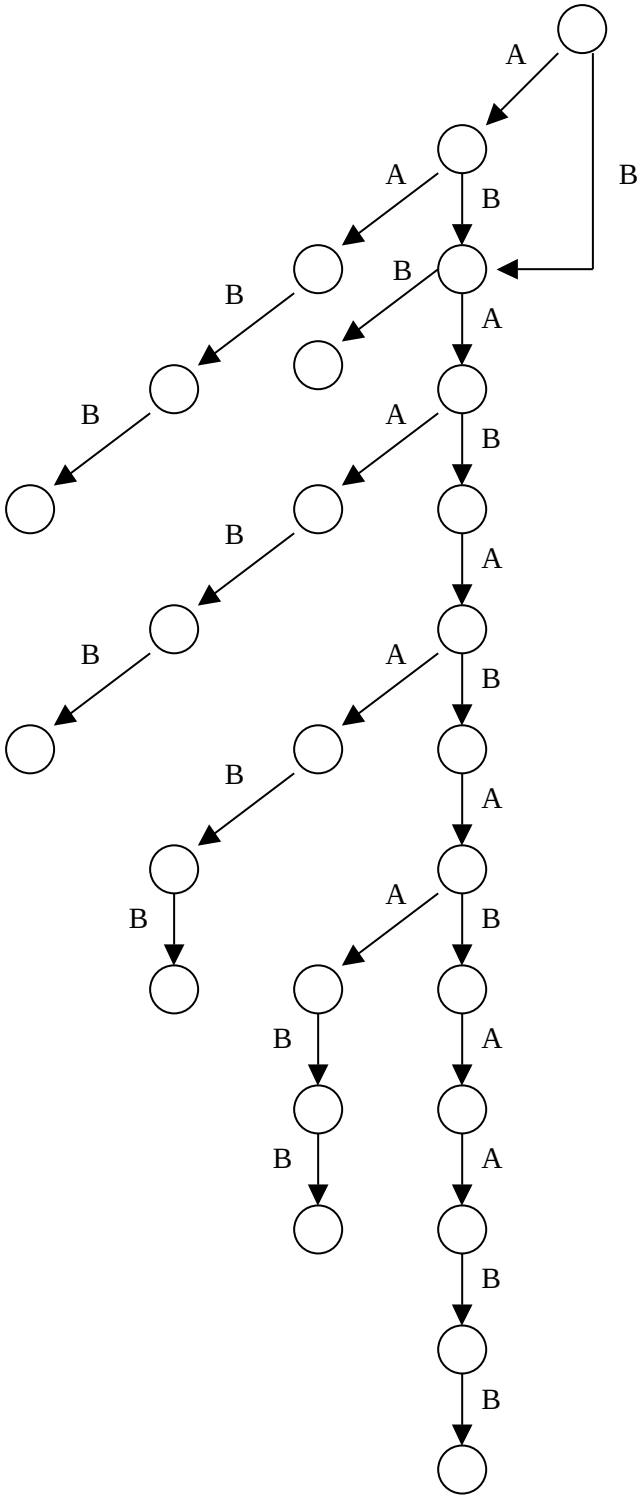


Fig. 9 : Construction de l'Oracle à partir de l'arbre suffixe du mot « ABABABABAABB », 1e étape.

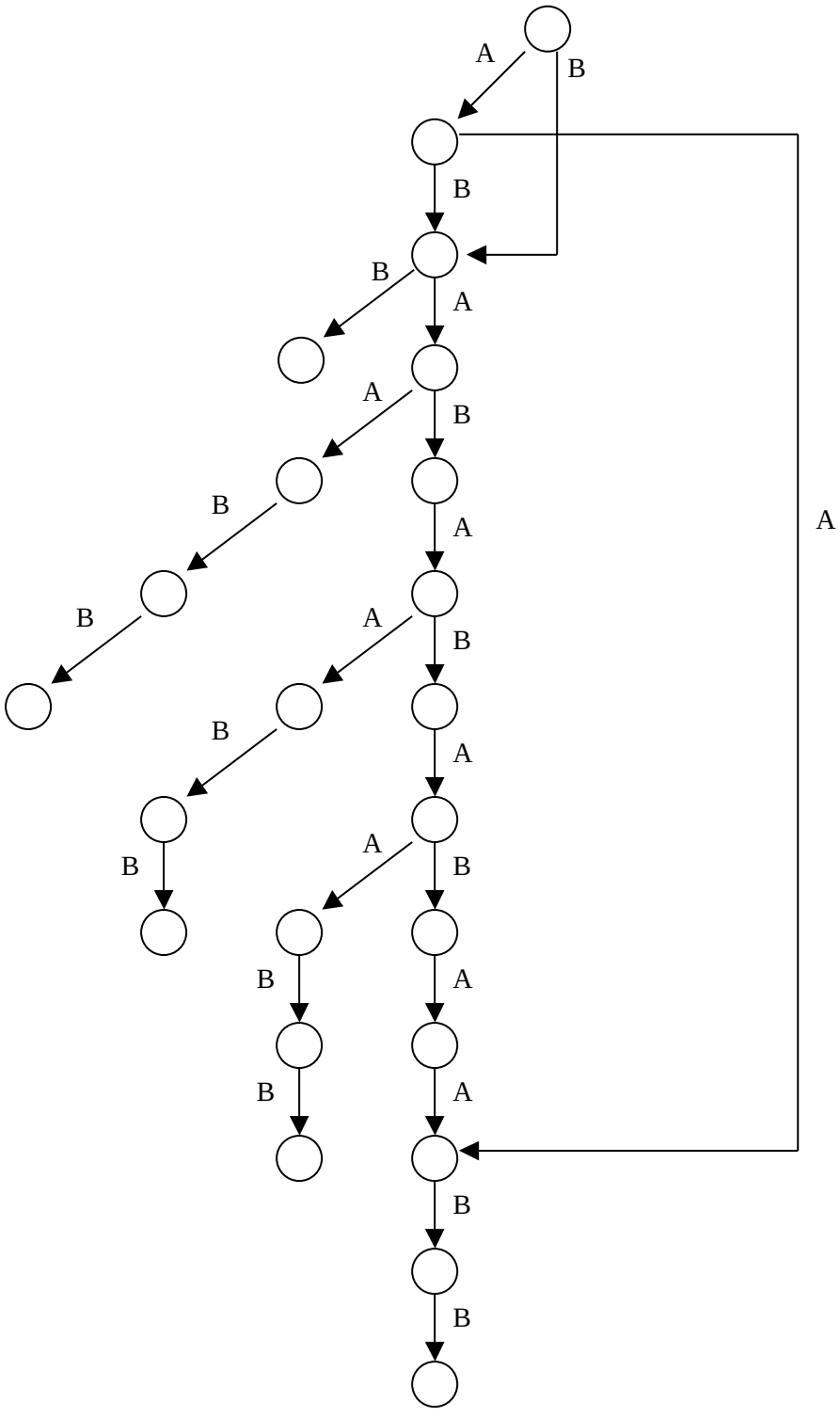


Fig. 10 : Construction de l'Oracle à partir de l'arbre suffixe du mot « ABABABABAABB », 2e étape.

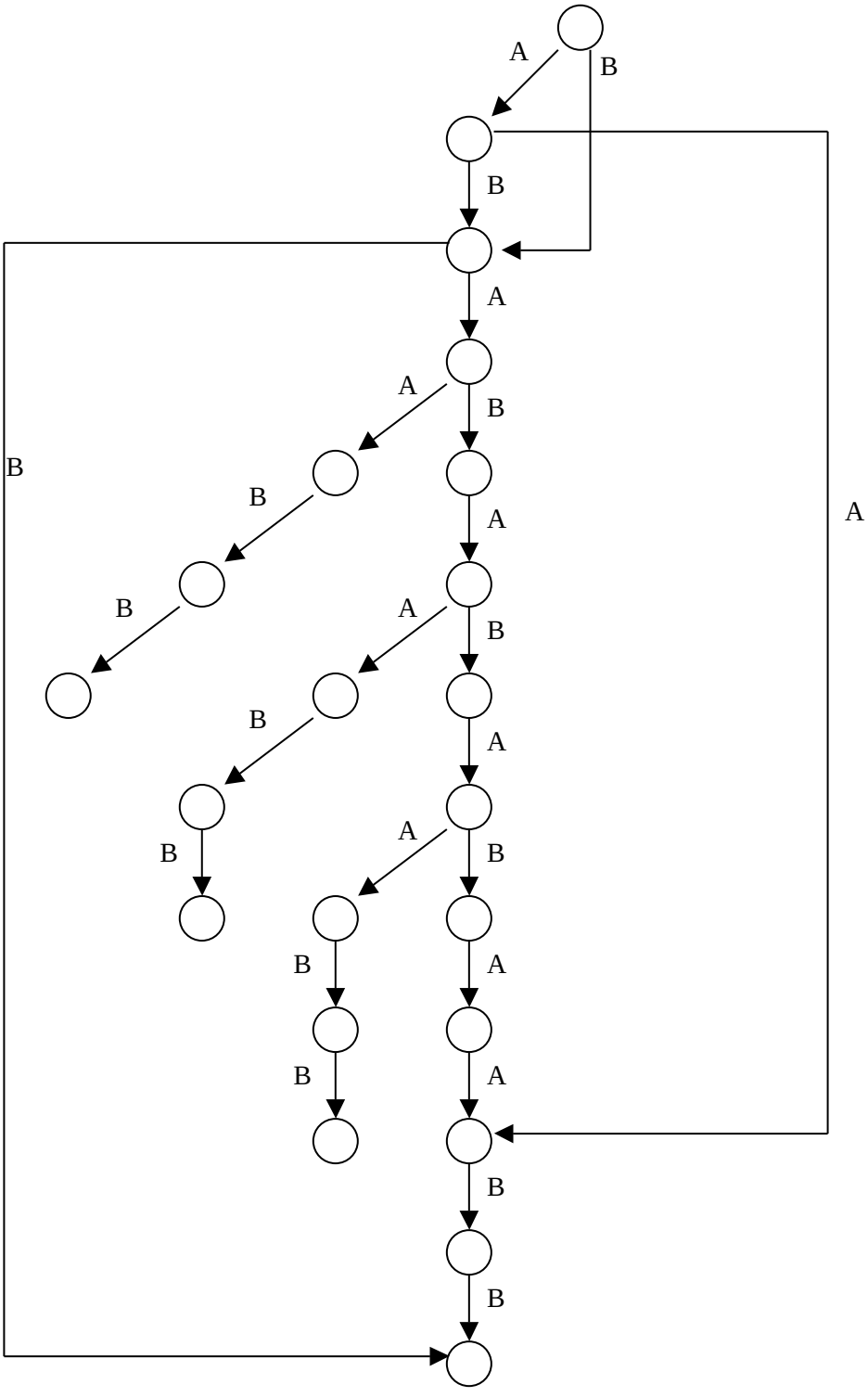


Fig. 11 : Construction de l'Oracle à partir de l'arbre suffixe du mot « ABABABABAABB », 3e étape.

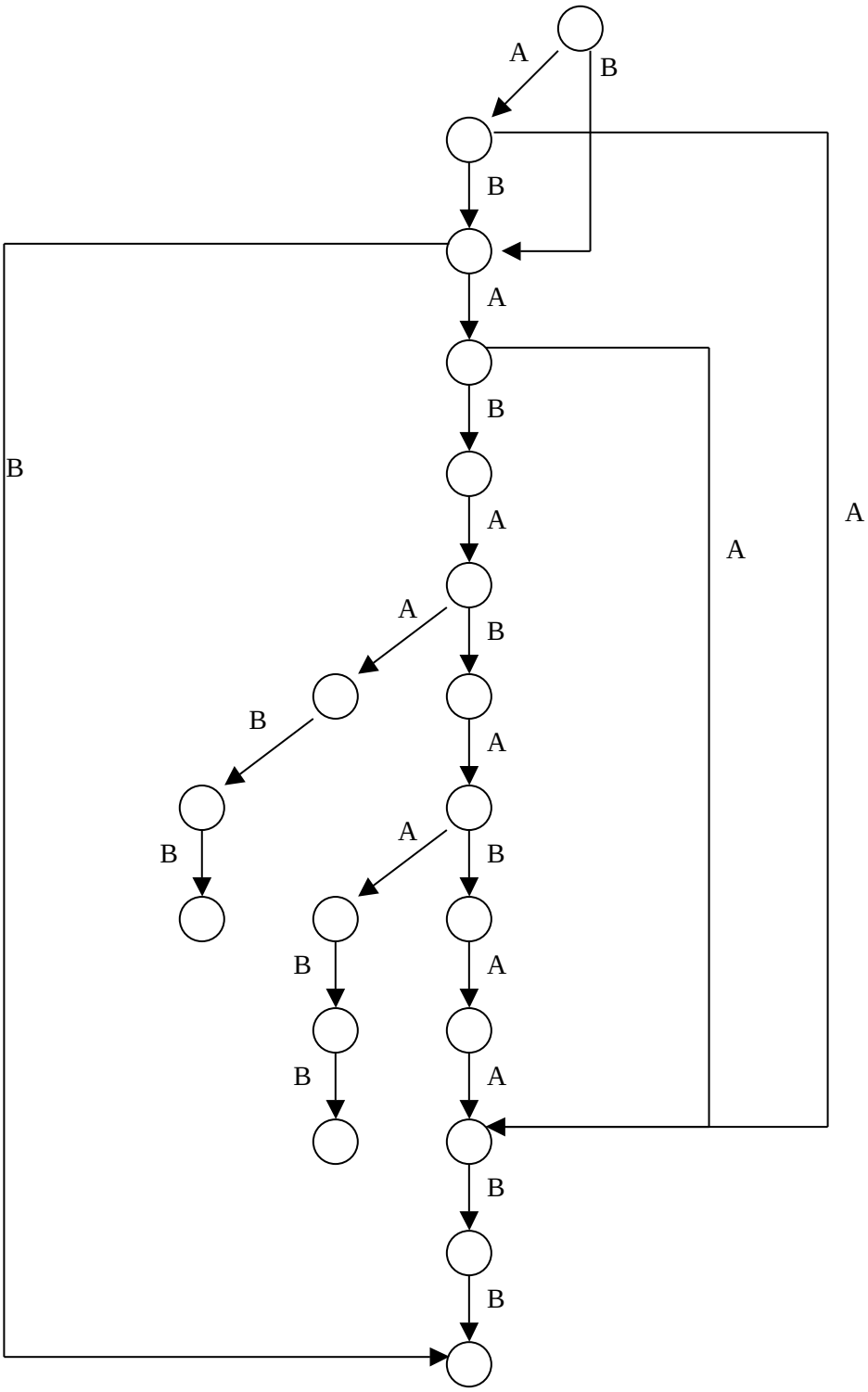


Fig. 12 : Construction de l'Oracle à partir de l'arbre suffixe du mot « ABABABABAABB », 4^{e} étape.

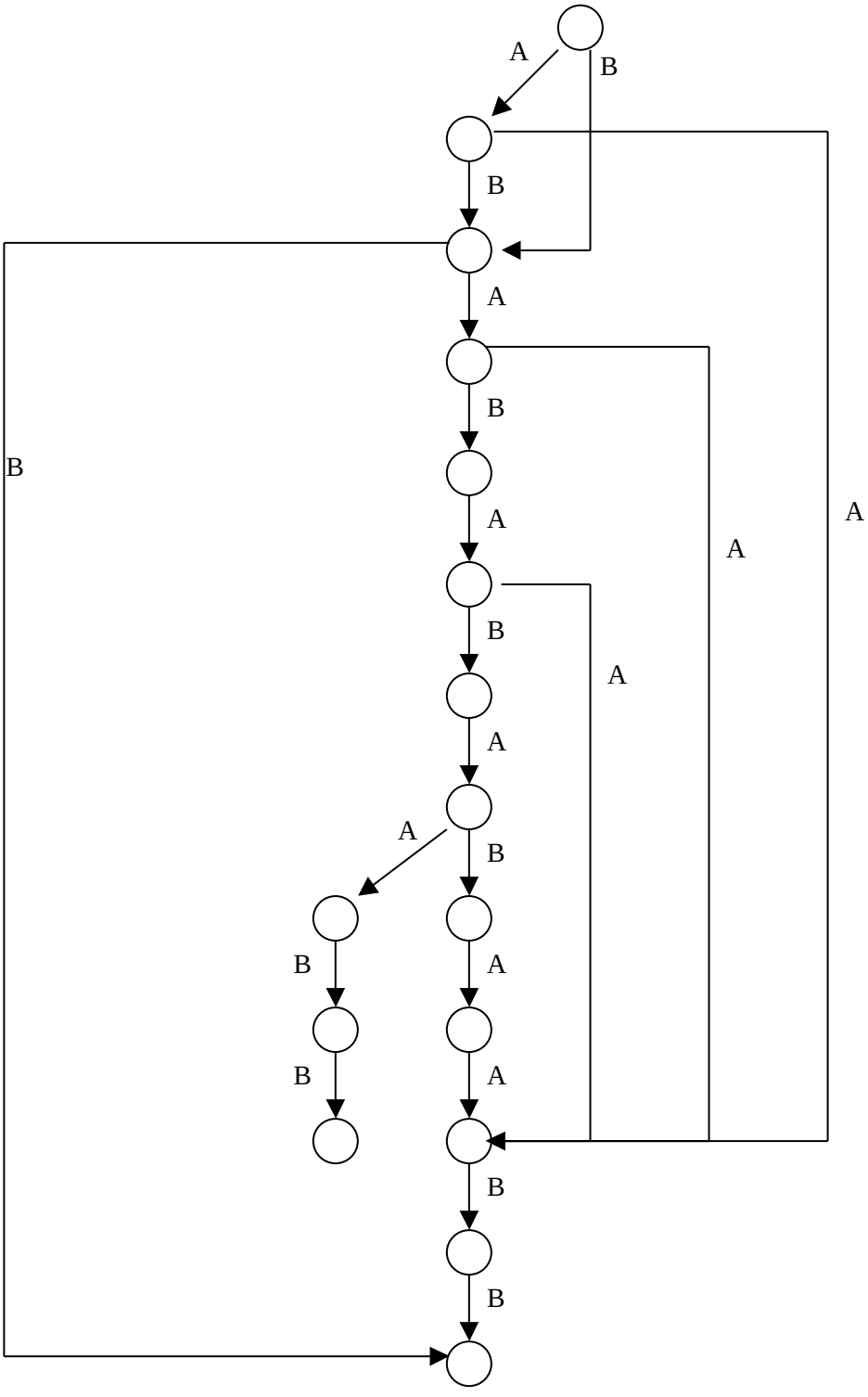


Fig. 13 : Construction de l'Oracle à partir de l'arbre suffixe du mot « ABABABABAABB », 5^{e} étape.

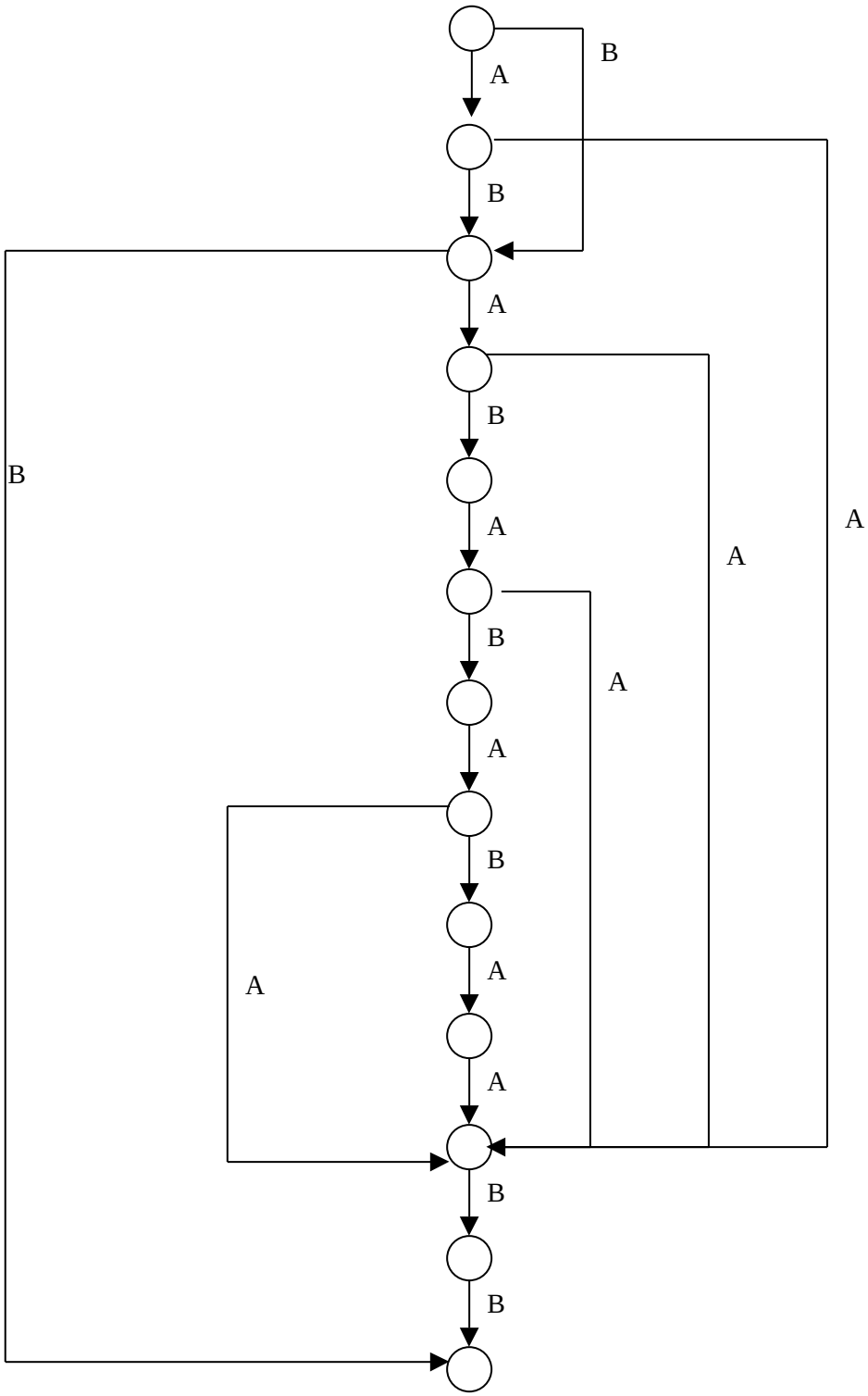


Fig. 14 : Oracle du mot « ABABABABAABB » construit à partir de son arbre de suffixes.

III.2.5 Vecteur « length repeated suffix » (LRS)

Lefebvre et Lecroq présentent dans [2] une méthode servant à calculer, grâce à un oracle des facteurs, la longueur des suffixes répétés au sein d'une séquence. On obtient un vecteur « length repeated suffix » qui à chaque état i associe la longueur du suffixe commun de p[1…i] et p[1…Sp(i)]. La méthode est linéaire en temps et en mémoire, et s'insère aisément dans l'implémentation de l'oracle.

On a vu l'utilité que peut avoir le vecteur de suppléances dans notre cas, liant entre eux des états en lesquels s'achèvent des facteurs redondants. Utiliser les liens suffixes lors d'un parcours du graphe élargit donc le choix des continuations possibles. Néanmoins, comme on en a fait la remarque précédemment, on ne connaît pas la longueur de ces facteurs communs. Autrement dit, si on utilise les liens suffixes lors du parcours du graphe, on ne sait pas de quelle taille est le contexte utilisé.

Construction du vecteur lrs

La méthode proposée dans [2] pour calculer la longueur du suffixe commun de p[1…i] et p[1…Sp(i)] est une récurrence que l'on peut mener parallèlement à la construction de l'oracle. Il s'agit de trouver un « ancêtre » commun, via les liens suffixes, au dernier état construit i et à l'état Sp(i+1)-1. On pourra se reporter à [2] pour la preuve de l'algorithme.

Lors de la construction de l'oracle du mot p[1…i+1] à partir de celui du mot p[1…i] et de la lettre p[i+1], les transitions arrière par le chemin suffixe s'arrêtent lorsqu'on atteint un état j tel que la transition (j,p[i+1]) est déjà définie. Le fait que cette transition existe déjà assure qu'il existe un état du chemin suffixe SPp(Sp(i+1)-1) dont la suppléance est j, à moins que $Sp(i+1) - 1 = j$.

On note π_1 l'état du chemin suffixe Sp(i) tel que $Sp(\pi_1) = j$.

Si $Sp(i+1)-1 \neq j$,
on note π_2 l'état du chemin suffixe SPp(Sp(i+1)-1) tel que $Sp(\pi_2) = j$.

Sinon (dans le cas où $Sp(i+\&) - 1 = j$), on note $\pi_2 = j$.

Ces notations données, la construction du vecteur lrs s'opère comme suit :

$$lrs(i+1) = \begin{cases} 0 & \text{si } Sp(i+1) = 0 \\ lrs(\pi_1) + 1 & \text{si } \pi_2 = Sp(\pi_1) \\ \min(lrs(\pi_1) , lrs(\pi_2)) + 1 & \text{sinon} \end{cases}$$

Et on initialise le vecteur par lrs(0) = 0.

On a dans [2] la preuve que lrs(i) est bien la longueur du suffixe commun de p[1…i] et p[1…Sp(i)]. Notons que ce *n'est pas* la longueur du facteur redondant maximal Repet(p,i). Cette « erreur » est due au fait que le lien suffixe (Sp(n)) est égal à *l'état en lequel Repet(p,i) est reconnu*, mais ne correspond pas nécessairement à une de ses occurrences.

On peut néanmoins dire que p[1…i] et p[1…Sp(i)] ont un suffixe commun non trivial (i.e. un caractère au moins). C'est la longueur de ce suffixe que l'on obtient avec le vecteur lrs.

III.2.6 Modalités de navigation

Le vecteur de lrs nous fournit, pour chaque transition suppléance, la taille du contexte commun aux deux continuations situées à chacune de ses extrémités. Dans le cadre de notre parcours de graphe, lorsqu'on cherche à bifurquer, on pourra ainsi avoir le choix entre l'ensemble des états du chemin suffixe généralisé (chemin suffixe exploré dans les deux sens), en sachant pour chacun des états quelle taille de contexte il respecte. On peut ainsi mettre sur pied une stratégie de « contexte maximal », permettant de choisir la continuation qui a le plus de cohérence par rapport au parcours déjà effectué dans le graphe.

La génération de séquences à partir de l'oracle est obtenue par un parcours partiellement stochastique. Pour conserver une certaine cohérence dans les séquences générées, on force ce parcours à utiliser les transitions directes (i.e. de type (n , σ , n+1)) un nombre de fois minimal avant de pouvoir bifurquer. Le paramètre réglant cette « continuité imposée » est stocké dans le champ **Max-Continuity** des objets de la classe Oracle dans Open Music. On agence donc des séquences de longueur Max-Continuity au minimum.

Lorsque enfin on offre à la navigation la possibilité de bifurquer par un lien suffixe, on recense tous les états de l'oracle que l'on peut atteindre par lien suffixe avant ou arrière, en utilisant éventuellement plusieurs liens suffixe à la suite. Il s'agit donc de lister tous les états situés sur des chemins suffixes dont fait partie l'état courant. On peut alors effectuer un tirage au sort, ou, utilisant le vecteur de lrs, choisir la transition qui assure un contexte de taille maximale.

Le choix de l'état de départ se fait par tirage au sort. On peut effectuer un parcours de l'oracle plus contraint (cf. VI.3.2, le cas du mode « beat » dans Omax), en imposant que la séquence générée respecte certaines caractéristiques. En l'occurrence, en mode « beat », les objets stockés dans l'oracle possèdent des labels. On veut que la séquence générée respecte une séquence de labels donnés. En ajoutant cette contrainte, on restreint les choix de transitions, mais on s'autorise à placer des objets vides dans la séquence générée lorsqu'aucun choix valide n'est disponible.

IV. ARCHITECTURE DU SYSTÈME OMAX

L'objet de l'expérience « Omax » est de faire interagir un musicien avec un système capable d'apprendre à l'imiter. Les deux modes d'utilisation d'Omax sont qualifiés de « free » et « beat ». Le mode free est celui de l'improvisation totalement libre, où l'on veut que la machine soit capable de réaliser l'acquisition de matériau musical et de restituer des simulations d'improvisation pendant que le musicien joue seul. Le mode beat, beaucoup plus contraint, installe une situation de temps-réel, où le système restitue un accompagnement construit autour d'une grille d'accords de jazz, avec un tempo et des harmonies « imposés ». Comme en mode free, le système « écoute » le musicien, emmagasine ce qu'il joue sur les harmonies de la grille d'accords, et restitue des improvisations synthétisées à partir de ce matériau.

Le problème des échelles temporelles multiples est déterminant pour la mise en place d'un tel système. En effet, on cherche à satisfaire des contraintes relevant à la fois du temps-réel (défilement métronomique de la grille d'accords en mode beat) et du temps différé (apprentissage de l'oracle, et temps compositionnel de la génération d'improvisations).

Omax est un système utilisant parallèlement MAX/MSP et Open Music, afin de profiter des avantages de chacun. Open Music est particulièrement adapté au traitement symbolique du matériau musical. MAX/MSP permet quant à lui l'interaction en temps réel avec la machine. Pour Omax, MAX/MSP fournit l'interface par laquelle l'instrument midi est connecté à l'ordinateur.

Les représentations informatiques et musicales utilisées déterminent grandement les échelles temporelles mises en jeu. Leur choix doit être adapté à la problématique posée. Pour nous, il s'agit principalement de trouver le grain de synchronisation entre l'échelle d'analyse du matériau musical et l'échelle de synthèse de l'improvisation. On discutera ce problème particulier lors de la description des modes d'interaction proposés dans Omax.

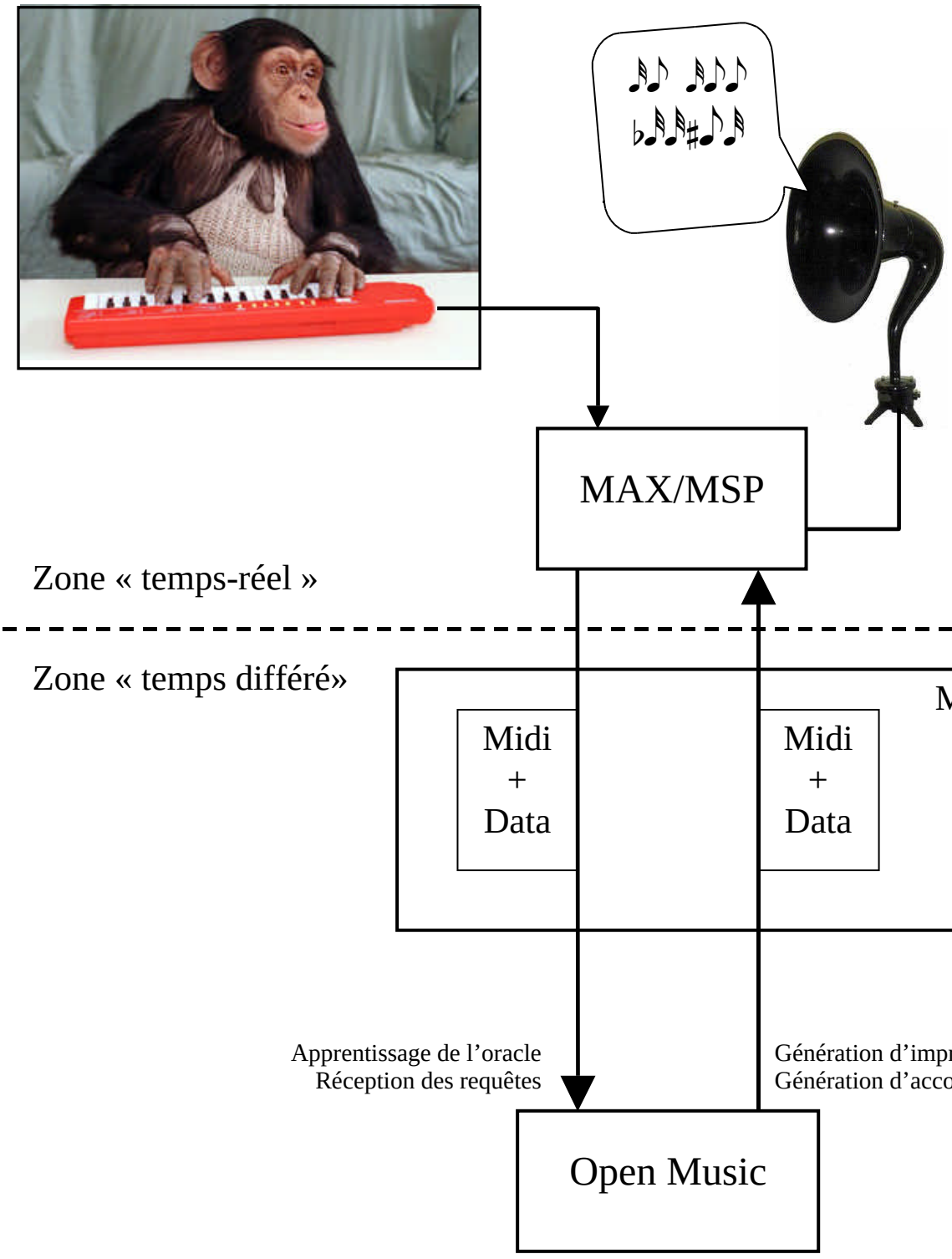


Fig. 15 : Architecture générale d'Omax

V.MAX/MSP ET MIDISHARE

Le travail mené au cours de ce stage a consisté principalement en l'implémentation de nouvelles structures dans la partie Open Music d'Omax. On se contentera donc dans cette partie de décrire l'interfaçage et les principales fonctionnalités de la partie MAX/MSP d'Omax. Elle consiste en des patches développés par M. Chemillier, dédiés aux différents modes d'interaction. S'il y a un seul patch pour le mode free, chaque grille d'accords du mode beat possède un patch dédié. En effet, outre l'interfaçage entre l'instrumentiste et Open Music, MAX/MSP est chargé de la réalisation de l'accompagnement. Il s'agit donc de diffuser simultanément des boucles audio (fichiers .wav), de lire des fichiers MIDI par le biais d'un expandeur en software, en respectant leur synchronisme.

Pour ce qui est de la communication avec Open Music, on utilise Midishare, qui ouvre des canaux de communication entre applications. Ainsi, ce qui est joué par l'instrumentiste est capté par MAX en envoyé dans un buffer Midishare, avec le tempo donné par le métronome. Lorsque l'instrumentiste arrête de jouer pendant un certain temps, le flux midi est coupé, et ce que contient le buffer transmis à Open Music pour l'apprentissage de l'oracle. Ce seuil d'inactivité est, par défaut, réglé sur 500 ms.

Midishare est aussi utilisé de façon détournée pour transmettre des instructions à Open Music. Ainsi, avant la fin de chaque grille, MAX effectue une requête à Open Music, lui demandant de générer une grille d'accords substitués (application de règles de substitutions à la grille de référence, cf. [7] et VI.3.2), et une improvisation de la durée d'une grille à partir du matériau assimilé par l'oracle. Cette improvisation est lue pendant la grille suivante. On peut alors utiliser deux modes d'interaction entre l'improvisateur-oracle et l'instrumentiste :

_ Un mode alternant où l'improvisateur-oracle se « tait » dès que l'instrumentiste joue. Il s'agit simplement d'un contrôle (mute / unmute) du volume du canal sur lequel est lu l'improvisation de l'oracle. On met ainsi en place un jeu de questions-réponse entre l'instrumentiste et la machine.

_ Un mode parallèle où l'improvisateur-oracle et l'instrumentiste jouent simultanément. Cela peut permettre à ce dernier de rebondir sans cesse sur ce que l'oracle joue, tout en continuant à lui fournir du matériau.

Les principaux contrôles disponibles sur les interfaces MAX/MSP pour Omax sont :

_ Une mixette permettant de balancer les niveaux sonores de l'instrumentiste, de la simulation d'improvisation, et des différentes parties de l'accompagnement en mode beat (boucles lues par MAX/MSP et flux midi générés par Open Music).

_ Un contrôle du tempo en mode beat.

_ Un contrôle du degré de substitution subi par la grille d'accords de référence en mode beat.

_ Le choix des boucles utilisées pour l'accompagnement en mode beat.

_ Des contrôles de l'oracle : Apprentissage en temps-réel on/off. Choix du mode d'interaction, choix de l'oracle dans la structure d'orchestre (oracles multiples). Choix de la zone de départ du parcours du graphe parmi l'ensemble du matériau acquis en mode free.

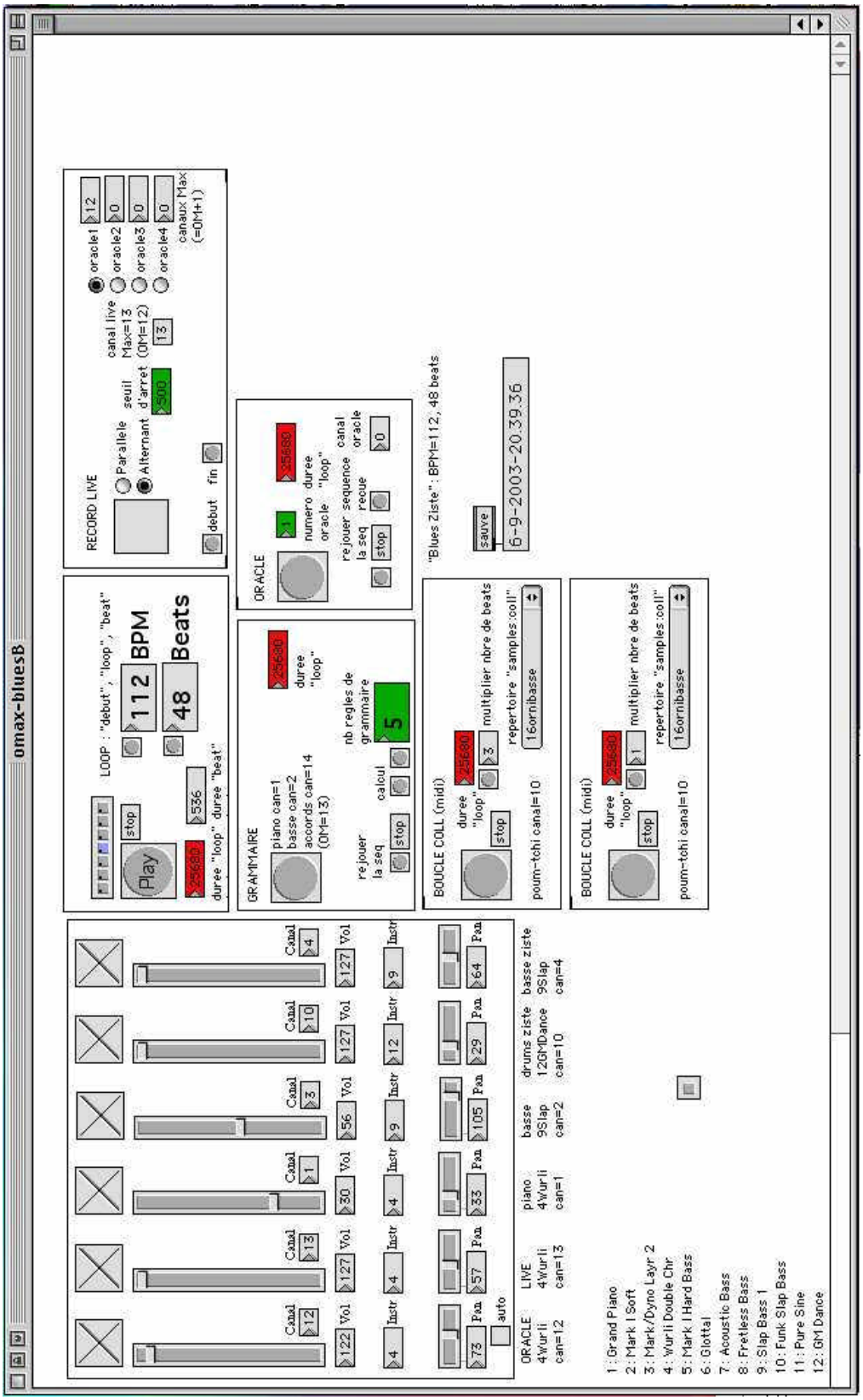


Fig. 16 : Interface MAX/MSP pour Omax en mode beat

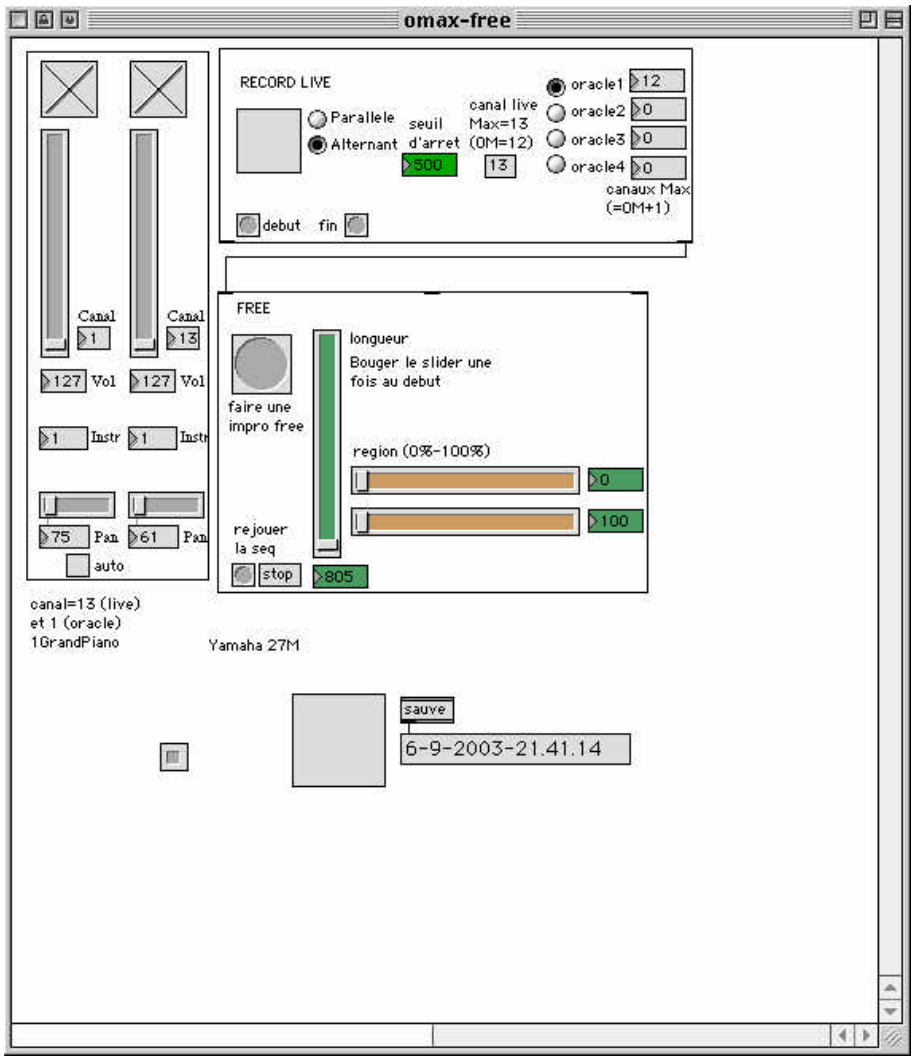


Fig. 17 : Interface MAX/MSP pour Omax en mode free

VI. L'ORACLE DES FACTEURS DANS OPEN MUSIC

VI.1. La classe oracle

E. Poirson donne dans [4] des détails de l'implémentation de l'oracle des facteurs dans Open Music. Comme évoqué précédemment, l'oracle conserve son caractère universel dans cette implémentation. Autrement dit, on peut utiliser la structure d'oracle pour organiser tout type de données : dans notre cas, ce seront des structures musicales détaillées dans le chapitre suivant.

Penchons-nous, pour commencer, sur les champs de la classe d'objets « oracle ». On ne décrira que les champs effectivement utiles actuellement.

_ Vectext est le champ contenant le « texte » traité. C'est un vecteur contenant la séquence « apprise ».

_ Hashtransition est un tableau contenant les transitions de l'oracle en tant que graphe.

_ Hashsuppl est un tableau contenant les transition suppléances. Utilisé à la création de l'oracle, on s'en sert aussi au moment de la génération de séquences, comme d'un jeu de transitions supplémentaire.

_ Hashsuppl-> est un tableau contenant les transitions suppléances *en sens inverse*. Cela nous permet d'augmenter les possibilités de bifurcation lors du parcours du graphe.

_ Veclrs est le vecteur contenant la valeur de lrs pour chaque état. Comme on l'a dit précédemment, cela nous donne accès à la taille du contexte pris en compte lors d'une bifurcation par une transition suppléance.

_ Maxetat est le nombre d'états que contient l'oracle, c'est-à-dire le nombre d'objets qu'on lui a fait assimiler.

_ Comparateur est un champ contenant le nom de la fonction de comparaison que l'on utilise lors de l'apprentissage. Par exemple, si on veut utiliser l'oracle sur un texte, on utilisera la fonction d'égalité sur les lettres. Pour des objets plus sophistiqués, on pourra utiliser des fonctions de comparaison plus particulières, adaptées aux besoins spécifiques de l'implémentation.

_ Max-continuity définit, par un entier, la taille minimale des séquences jouées sans utiliser de lien suffixe pour bifurquer lors du parcours du graphe. Autrement dit, lors de la génération, l'algorithme agence des séquences de taille Max-continuity au minimum.

_ Start-region contient, en %, les frontières de la zone où l'on désire que commence le parcours du graphe lors d'une génération. Par exemple, pour que la génération commence à partir du matériau assimilé le plus récemment, on utilisera (95 100) (début de la génération entre 95% et 100%, c'est-à-dire parmi les derniers 5% de matériau entrés).

_ Fwsuffix, Bwsuffix sont deux booléens indiquant si l'on veut utiliser les liens suffixes et/ou les liens suffixes arrières. Dans l'état actuel de l'expérience, on utilise toujours ces deux options, qui multiplient les possibilités de bifurcation.

_ Bestsuffixmode est un booléen indiquant si, au moment d'une bifurcation dans le parcours du graphe, on choisit (grâce au vecteur de lrs) toujours la transition garantissant un contexte maximal.

_ RefHamrgrid contient, dans le cas de notre expérience avec accompagnement, la grille d'accords de référence.

VI.2. Fonctions de base de la classe oracle

_ Initialize-instance : création et initialisation d'une instance de la classe Oracle.

_ Ajouter-objet : Fonction *générique* d'ajout d'un objet à l'oracle. Elle fait appel à la fonction de comparaison dont le nom se trouve dans le champ « comparateur » de l'oracle.

VI.3. Représentations informatiques

VI.3.1 Mode Free

Le premier type de protocole que nous présentons est basé sur l'improvisation au sens le plus large et libre qui soit. On propose à un instrumentiste seul d'improviser, sans contrainte de tempo ou d'harmonie. L'idée est d'utiliser les représentations adéquates du matériau musical pour, tant que faire se peut, offrir à l'instrumentiste de jouer avec un clone numérique. L'effet recherché est une captation de son mode de jeu, en particulier du point de vue des profils de dynamique.

On comprend que, toute liberté étant donnée au musicien quant au tempo ou à la « densité » de son improvisation, l'échelle temporelle d'analyse de ce processus est grandement dépendante de son jeu. Aussi la granularité utilisée pour stocker ces données dans l'oracle est-elle la plus petite possible.

O. Lartillot [3] soulève le problème de « représenter un discours musical sous la forme d'une séquence de symboles de telle manière qu'il y ait une équivalence entre cette représentation séquentielle et la représentation initiale du discours musical ». La réponse à cette problématique (Assayag, Dubnov, Delerue, [6]) est implémentée dans Open Music sous la forme d'une classe d'objets simples, les *CrossEvents*. L'idée est d'utiliser un « Cross alphabet » de symboles de la forme

({(hauteur1,vélocité1) , … , (hauteurN,vélocitéN)}, durée)

En l'occurrence, le flux de données midi reçu de l'instrumentiste est découpé en segments élémentaires dans Open Music. Chaque événement midi (« note on » ou «note off ») donne lieu à une coupure du flux. On obtient ainsi une représentation complète et exacte du flux midi sous la forme d'une concaténation de CrossEvents. Ceux-ci contiennent une durée et un jeu de données midi.

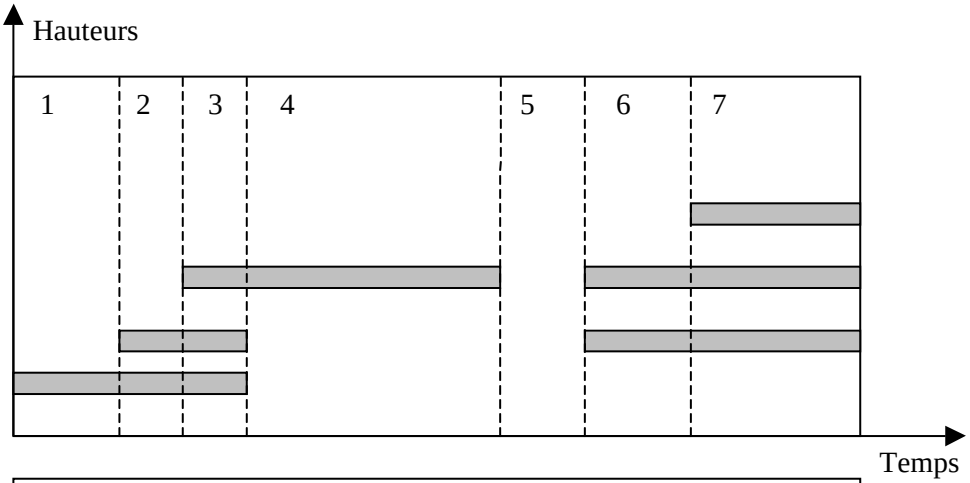


Fig. 18 : Sectionnement d'un flux de données midi en tranches polyphoniques de type CrossEvents.

Une fois cette représentation formulée, il reste à définir un mode de comparaison des CrossEvents. Pour utiliser la structure d'oracle, il nous faut en effet convenir d'une fonction de comparaison des objets contenus par l'oracle, sur laquelle sera fondée la caractérisation des redondances dans le matériau musical. On a besoin d'établir une distance booléenne entre les CrossEvents, puisqu'il s'agit de déterminer si deux CrossEvents peuvent être « musicalement confondus ». E. Poirson [4] a pour cela exploré diverses possibilités, en particulier en ce qui concerne la comparaison des accords.

La fonction de comparaison retenue pour le mode free consiste en :

- Vérifier si les CrossEvents possèdent le même jeu de hauteurs,
- Sinon, s'ils possèdent chacun strictement plus de 2 notes, mesurer leur distance Estrada (distance des textures des accords, travail de G. Bloch sur un algorithme du musicologue J. Estrada),

· Sinon, toujours s'ils possèdent chacun strictement plus de 2 notes, comparer leurs fondamentales absentes par une méthode du compositeur et musicologue Hindemith.

On note que la durée des CrossEvents n'intervient pas dans ce processus de comparaison exclusivement harmonique.

Champs de la classe « CrossEvent »

Cette classe hérite de la classe « Event », et possède strictement les mêmes champs.

_ Duration est la durée de cette tranche polyphonique en millisecondes.

_ MidiSet contient le jeu d'informations midi de l'event. Celles-ci sont de la forme (pitch onset durée velocité canal). Dans le cas des CrossEvents, on a toujours onset = 0 et durée = Duration.

VI.3.2 Mode Beat

Le « mode beat » est destiné à des musiciens dont l'approche de l'improvisation est liée au jazz. Ce mode, bien plus contraint que le mode free, impose un accompagnement à l'improvisateur : le tempo est fixe, la basse, la batterie et le clavier générés automatiquement jouent un accompagnement autour de la grille d'accords de référence du morceau. Une grammaire de substitution d'accords tirée de Steedman [7] et implémentée par M. Chemillier sous Open Music permet de générer un accompagnement sans cesse nouveau autour de la grille de référence.

Ce que l'on va chercher à analyser dans ce mode d'improvisation est la corrélation existant entre le jeu de l'improvisateur et les harmonies sous-jacentes, imposées par la grille d'accords de référence du morceau. L'aptitude caractéristique des improvisateurs jazz est celle de développer (au sens compositionnel) leurs idées musicales immédiates sous la contrainte des harmonies de l'accompagnement. On comprend bien que le modèle de type « free » présenté précédemment ne peut pas convenir pour ce type d'expérience. En effet, on y perd la corrélation entre le matériau et la grille d'accords. Les effets de modulation et de suivi de l'harmonie ne pourraient pas être imités.

Le modèle retenu en mode beat possède un grain d'analyse bien plus large que dans le mode free. Les objets que l'on utilise pour l'apprentissage de l'oracle représentent un beat de musique, au tempo imposé par l'accompagnement. Le flux musical est donc découpé suivant le métronome. Les beats sont labellisés par l'accord sur lequel ils ont été joués, et leur comparaison s'effectue exclusivement sur les labels harmoniques. La génération d'improvisation est à son tour contrainte par la grille d'accords de référence ; on obtient une recombinaison de larges morceaux de l'improvisation originale. En ce sens, on peut dire que le mode beat consiste en un sampling de l'improvisateur humain : celui-ci pourra reconnaître des morceaux de ses phrases, recombinés par l'oracle. Il est donc nécessaire, si on veut profiter au mieux de l'innovation que peut fournir l'oracle, d'effectuer son apprentissage sur une grande quantité de matériau.

Grammaire de substitution des accords

M. Chemillier a implémenté dans Open Music un système de substitutions d'accords inspiré de Steedman [7]. Ce sont des règles de substitution, qui permettent de générer, à partir de la grille d'accords de référence, une grille d'accords la « paraphrasant ». Un tirage au sort des règles à appliquer ainsi qu'un contrôle du nombre d'itérations du processus permettent d'obtenir des grilles plus ou moins fidèles à la grille originale. Surtout, cela permet d'éviter la monotonie d'un accompagnement suivant immuablement la même grille.

La grammaire de substitution des accords réalise des sections de mesures, fait apparaître des cadences de type II V I, et effectue des substitutions tritoniques.

Pour présenter ci-dessous la grammaire utilisée, on pose les notations :

x est un degré sur lequel on effectue la substitution.
(x) est l'accord « x majeur ».
(x 7) est l'accord « x 7^e ».
(x m7) est l'accord « x mineur 7^e ».
/ est une barre de mesure.
w est un « joker » (n'importe quel accord).
(D x) est le degré « dominante de x ».
(Stb x) est le degré « triton de x ».

1-1 : (/ (x) /) -> (/ (x) (x) /)
1-2 : (/ (x 7) /) -> (/ (x) (x 7) /)
1-3 : (/ (x m7) /) -> (/ (x) (x m7) /)

3a-1 : (w (x 7)) -> (((D x) 7) (x 7))
3a-2 : (w (x 7)) -> (((D x) m7) (x 7))

3b-1 : (w (x m7)) -> (((D x) 7) (x m7))

4-1 : ((D x) 7) (x 7)) -> (((Stb x) 7) (x 7))
4-2 : ((D x) 7) (x)) -> ((Stb x) (x))
4-3 : ((D x) 7) (x m7)) -> ((Stb x) (x m7))

3a-11 : (w / (x 7)) -> (((D x) 7) / (x 7))
3a-21 : (w / (x 7)) -> (((D x) m7) / (x 7))

3b-11 : (w / (x m7)) -> (((D x) 7) / (x m7))

4-11 : (((D x) 7) / (x 7)) -> (((Stb x) 7) / (x 7))
4-21 : (((D x) 7) / (x)) -> (((Stb x) 7) / (x))
4-31 : (((D x) 7) / (x m7)) -> (((Stb x) 7) / (x m7))

La première réalisation d'Omax effectuait l'apprentissage des beats en les labellisant par les accords substitués sur lesquels ils avaient été joués. Lors de la génération d'improvisation, le parcours de l'oracle était contraint par les accords de la grille substituée suivante. Autrement dit, on réalisait l'apprentissage et la

génération sur un corpus d'accords bien plus grand que celui de la grille de référence. Ce protocole est tout à fait viable si l'improvisateur enregistre une grande quantité de matériau. Si certains accords d'une grille substituée sont « inconnus » de l'oracle (i.e. si aucun des beats qu'il contient n'est labellisé par ces accords), il laissera un beat « blanc » dans son improvisation.

Pour des raisons pratiques concernant les tests du système, ce procédé est actuellement désactivé. Les substitutions sont encore effectives pour l'accompagnement, mais les beats sont labellisés suivant la grille de référence. Les improvisations générées par l'oracle sont contraintes par cette même grille. Comme le système de substitutions ne fait que « paraphraser » la grille de référence, en itérant quelques fois seulement le processus de substitution, on obtient des grilles assez proches de celle de référence pour que soit justifiée cette simplification.

Champs de la classe « Beat »

Cette classe hérite de la classe « Event », et possède donc les mêmes champs. Elle possède des champs supplémentaires :

_ HarmLabel est le champ contenant le label harmonique du beat tel que transmis par MAX (issu de la grille d'accords substitués).

Ce label est codé par un couple (degré typeAccord) où « degré » est la fondamentale de l'accord, de 0 pour do à 11 pour si bémol, et « type accord » code le type de l'accord : 1 pour « majeur », 2 pour « mineur 7 », et 3 pour « 7^{e} de dominante ».

Comme expliqué précédemment, on a désactivé ce champ et on labellise les beats par l'accord de référence.

_ RefHarmLabel est le champ contenant le label harmonique de référence, obtenu à partir du numéro du beat au sein de la grille, et de la grille d'accords de référence mémorisée dans l'oracle. Le codage est décrit ci-dessus.

_ NumBeat est le numéro du beat dans la mesure.

_ StartPhrase est un booléen indiquant si le beat est le premier d'une séquence reçue par Open Music. C'est un marqueur de « début de phrase ».

_ QMidiSet est le midiset du beat *après quantification*.

_ RelHarmLabel est un champ codant la transition harmonique entre le label du beat et le label suivant dans la grille d'accords. Pour le codage, cf. paragraphe VII.2.2.

VII. VERS UN ORACLE INVENTIF

VII.1. Considérations temporelles

Le choix des échelles temporelles est, dans le cas de notre expérience, crucial. En effet, on cherche à faire collaborer des processus dont l'horizon temporel est foncièrement différent. Pour qu'on puisse parler d'interaction homme-machine, le temps réel est, dans notre cas, indispensable. On veut atteindre une transparence d'utilisation qui ne peut être garantie que par des latences de l'ordre de 10 ms. MAX/MSP est dédié à ce type d'utilisation, et fournit une base solide au système.

Cependant, l'apprentissage de l'oracle et la génération d'improvisations dans Open Music est réalisé en temps différé. Il ressort d'une discussion avec G. Bloch que l'on peut distinguer deux types d'échelles temporelles intervenant dans ces processus :

_ *Le grain d'analyse* des événements musicaux.

_ *Le « scope »*, horizon temporel de la génération d'improvisation, vue comme un processus de composition.

Ces deux échelles sont liées, puisque la génération d'improvisation est réalisée à partir du matériau assimilé par l'oracle. Il convient donc de choisir judicieusement le grain d'analyse, qui détermine grandement les caractéristiques du système final. On verra ci-après que le scope peut lui être contraint par le mode d'interaction choisi.

On a évoqué précédemment les deux modes disponibles actuellement dans Omax. Pour rappel :

_ Le mode free possède un grain d'analyse « atomique », constitué de tranches polyphoniques séparées par les événements midi « note on » et « note off ». Ce grain est irrégulier, déterminé par le jeu de l'instrumentiste. En ce qui concerne le scope de ce mode, on peut demander à l'oracle de générer une improvisation d'une longueur arbitraire. Il s'agit néanmoins de limiter celle-ci de façon à ce que le système puisse prendre en compte le matériau nouveau le plus rapidement possible.

_ Le mode beat possède un grain d'analyse plus large, déterminé par le métronome. Ce grain est régulier et indépendant du jeu de l'instrumentiste. La réactivité que l'on exige du système nous contraint à faire générer par l'oracle des

improvisations de la durée d'une grille d'accords. Le scope de ce mode est particulièrement contraint par la durée de la grille d'accords. Ce scope nous interdit d'avoir un contrôle plus large, par exemple de pouvoir générer une improvisation construite sur trois grilles d'accords avec un début, une progression et une conclusion. En effet, comme en mode free, un élargissement du scope ralentit la réactivité du système au nouveau matériau joué par l'instrumentiste.

VII.2. Un oracle « hybride »

Quel que soit le mode utilisé dans Omax, les improvisations générées sont une recombinaison des objets contenus par l'oracle. On possède un contrôle sur la taille minimale des séquences que l'oracle recombine, et les tests montrent effectivement que l'on peut reconnaître des passages de l'improvisation originale. En mode free, le grain d'analyse est assez fin pour donner l'illusion d'une réappropriation du matériau par la machine.

En revanche, en mode beat, ce sont des extraits de l'improvisation originale, d'une durée d'un beat, qui sont recombinés. Leur régularité métronomique renforce l'impression que ce sont des samples accolés en une séquence musicale. De plus, il se pose le problème du contenu musical des beats. Comme l'oracle ne les recombine que d'après leur marqueur harmonique, il est susceptible de générer des phrases musicales « bancales » en comparaison avec ce que peut jouer un musicien humain. Cela peut être dû à des cassures rythmiques aussi bien que mélodiques.

Il nous a semblé, sur la base de ces remarques, que le rythme et le phrasé dans le style d'improvisation étaient jusque-là négligés alors qu'ils participent fortement du réalisme d'une simulation d'improvisation. Les représentations utilisées pour le mode free permettent à l'oracle de reproduire de façon efficace des profils mélodiques de l'improvisateur humain. Il est légitime de supposer que, de la même façon, on pourrait faire reproduire à un oracle les combinaisons rythmiques qui sous-tendent le phrasé d'un musicien.

L'approche la plus simple pour tester la validité de ces suppositions a été de détourner le mode beat de Omax, afin de créer un oracle rythmique. Par la suite, il est apparu nécessaire de développer séparément un oracle mélodique qui, en complément de la génération d'une phrase rythmique, puisse engendrer une mélodie à y apposer.

De cette façon, en décorrélant la génération rythmique et la génération mélodique, on s'assure une bien plus grande inventivité de la part de l'oracle. Mais on peut se demander quelle trace on garde du style original de l'instrumentiste, puisque le procédé de « recollement » des rythmes et des mélodies générées participe fortement du résultat final.

VII.2.1 L'oracle rythmique

L'efficacité mélodique de l'oracle free montre la nécessité de choisir avec soin le grain d'analyse en fonction du problème. Pour obtenir un oracle rythmique pertinent, il paraît nécessaire de pouvoir manipuler des objets rythmiques symboliques, de plus haut niveau que les simples durées des CrossEvents. C'est la raison pour laquelle cet oracle rythmique a été développé sur la base du mode beat de Omax.

En effet, dans le cadre de l'improvisation jazz, l'improvisateur développe une capacité d'improvisation rythmiquement très contrainte par le tempo qui met en valeur les patterns rythmiques. En s'appuyant sur le découpage métronomique du flux musical, on peut extraire de façon simple les patterns rythmiques symboliques à l'échelle d'un beat. Leur séquencement est quant à lui représentatif du phrasé du musicien.

L'idée présidant à la création de l'oracle rythmique est d'utiliser une fonction de comparaison des beats fondée sur les patterns rythmiques. On utilise, pour coder les patterns rythmiques, le formalisme des *arbres rythmiques et des ratios* existant déjà dans Open Music. On désire en un sens que les arbres rythmiques définissent des classes de patterns dont les réalisations peuvent admettre un certain nombre d'imperfections.

A ce propos, citons R. Rowe, [9], chapitre « Expressive performance » : « L'exécution par un séquenceur d'une partition quantifiée donne un résultat musical « sans vie », comme peut en témoigner quiconque est un tant soit peu familier avec les séquenceurs. Les musiciens humains commettent des déviations temporelles, de dynamique et timbrales à partir de la partition, afin d'exprimer une interprétation de la musique. Certaines de ces déviations sont inhérentes à différents niveaux de technique, ainsi qu'à de simples faiblesses de la musculature humaine. Néanmoins, les recherches des dernières décennies ont montré que la plupart de ces variations sont intentionnelles, et soigneusement contrôlées par les musiciens chevronnés (Palmer, 1989). ».

Notre credo consiste précisément à modifier le moins possible le matériau portant l'expressivité. Comme pour le mode beat d'omax, ce qui est généré par l'oracle rythmique n'est, du point de vue des onsets et des vélocités, pas quantifié, mais *rejoué tel quel*. On espère ainsi capter une part du style d'improvisation du musicien ayant réalisé l'apprentissage de l'oracle : le rythme de son phrasé.

Les rytbeats

On crée une nouvelle classe, étendant celle des « beats » utilisés jusque là. Ces « rytbeats » sont des objets construits à partir d'une séquence midi de durée un beat.

La classe « rytbeat » hérite ses champs de la classe beat. Les champs supplémentaires de la classe « rytbeat » sont :

_ NextHarmLabel est le label harmonique du beat suivant (cela sert à enrichir le contexte harmonique, cf. plus bas pour des précisions).

_ HarmTrans est un champ où est codée la transition harmonique entre RefHarmLabel et NextHarmLabel, de façon relative (cf. plus bas).

_ Rytree contient l'arbre rythmique décrivant la structure simplifiée du beat.

_ Ratio contient l'arbre rythmique sous forme d'un ratio, utile pour les comparaisons de rytbeats.

_ Pitchtree est la liste des pitchs des événements mentionnés dans l'arbre rythmique.

_ Duration contient la durée du beat d'après les tics du métronome, ce qui permet de vérifier la robustesse du système Midishare.

_ Eventlist contient la séquence de CrossEvents du beat.

_ Nbnotes contient la liste du nombre de notes simultanées dans chaque CrossEvent.

_ PreBeatMS contient le MidiSet d'événements situés *avant* le beat mais jugés lui appartenir (événements déclenchés dans les 10% de la fin du beat précédent).

_ PostBeatMS contient le MidiSet d'événements situés *après* le beat mais jugés lui appartenir (événements s'achevant dans les premiers 10% du beat suivant).

La méthode **make-rytbeat** permet de constituer un rytbeat à partir du jeu d'informations (LabelHarmonique Durée MidiSet). On utilise entre autres la méthode **midi->cross** développée par O. Lartillot (cf. [3]) pour transformer la séquence midi en une séquence de CrossEvents. Cela nous est utile pour analyser la structure rythmique du beat.

Les champs PreBeatMS et PostBeatMS sont utilisés pour stocker des morceaux de séquence midi qui, si l'on respectait précisément le découpage métronomique, appartiendraient au beat précédent ou suivant. Comme on recombine des beats lors de la génération, on risquerait de perdre des effets d'irrégularité caractéristiques d'un jeu humain. L'utilisation de ces champs découle de la constatation que ce qu'on associe, musicalement parlant, à un beat, peut effectivement en « dépasser ». Au moment de la génération d'une phrase rythmique, on recolle donc ces morceaux de séquence midi au midiset principal du rytbeat.

VII.2.2 Arbre rythmiques

Dans le but de comparer aisément des structures rythmiques à l'échelle du beat, on utilise le formalisme des arbres rythmiques. Les arbres rythmiques simples que nous utilisons se présentent sous la forme d'une liste d'entiers. La somme des valeurs absolues de ces entiers indique la profondeur de notre découpage rythmique. Ainsi, dans l'exemple figure 19, cette somme vaut 8, ce qui signifie que notre unité de base est le 1/8^{e} de temps, soit la double-croche. Les entiers de la séquence représentent alors une suite de durées figurées en cette unité. Les négatifs indiquent du silence. Si on rapporte la somme des valeurs absolues à 1, on obtient le *ratio de la séquence rythmique*. C'est sur la comparaison des ratios qu'est fondée la fonction de comparaison entre rytbeats **CompareRytbeats**.

Comme on cherche des formes rythmiques assez simples, uniquement à l'échelle du beat et qui plus est dans le cadre du style d'improvisation jazz, il semblait pertinent de mettre en place un procédé d'extraction simple et léger. Cette extraction est réalisée par la méthode **beatquantifybis**. On calcule une distance des on sets des événements de la séquence musicale à des peignes réguliers découpant le beat en 1, 2, 3, 4, 6 ou 8 parties égales. Après avoir décidé de la division du beat

à adopter, il est aisé de transformer la liste des durées des événements en arbre rythmique.

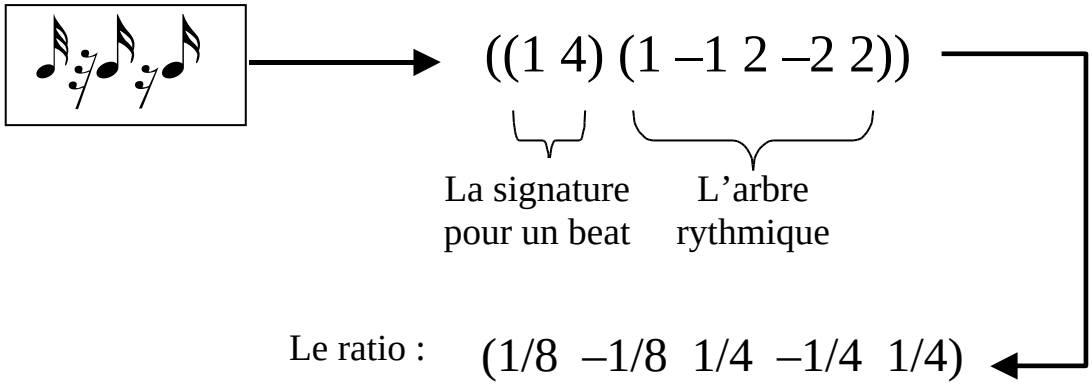


Fig. 19 : Transformation des informations midi d'un beat en arbre rythmique, et en ratio.

VII.2.3 Labels harmoniques relatifs

Les labels harmoniques tels qu'utilisés jusque là n'étaient que des marqueurs permettant à l'oracle de ne pas faire d'erreur grossière lors de la génération d'improvisation. Néanmoins, l'approche menée était totalement agnostique. Pour un label donné, au moment de la génération, l'oracle était simplement contraint à recombiner des beats joués sur ce label par l'instrumentiste, sans plus de connaissances musicales.

Notre approche se veut plus générale et musicale. En particulier, il est dommage que l'oracle ne puisse pas transposer les phrases mélodiques d'un label à un autre. Aussi a-t-il paru très avantageux de fournir à l'oracle cette capacité de transposition. L'idée est d'effectuer un stockage relatif du matériau musical.

Les *types d'accords* connus par l'oracle sont les accords majeurs, mineur 7 et 7^{e} de dominante. Le label harmonique est un couple (degré type-d'accord). On a donc implémenté une transposition au moment du stockage du matériau : le midiset de chaque beat est transposé en do avant apprentissage de l'oracle. Inversement, au moment de la transposition, il suffit d'effectuer la transposition retour vers la tonalité exigée par la grille.

Dès lors, l'oracle peut utiliser tout ce qui a été joué sur un type d'accord donné pour générer une séquence de beats sur ce type d'accord.

On conviendra que les connaissances musicales données là à l'oracle sont très réduites : il sait simplement opérer des transpositions basiques. Un problème subsiste : deux accords du même type n'ont pas nécessairement le même statut au sein d'une grille d'accords. Le fait que l'oracle ne possède pas de connaissances concernant la modalité des accords est gênant. En effet, lorsqu'un improvisateur humain prend soin de moduler différemment son phrasé sur les accords Dm7 et Em7 (tonalité C majeur), l'oracle risque lui de mélanger du matériau issu des deux labels pour improviser indifféremment sur l'un ou l'autre. Notre codage de labels harmoniques relatifs est donc de ce point de vue insuffisant, puisqu'il introduit une trop forte décorrélation entre le matériau musical et la grille d'accords sous-jacente.

On propose une solution à ce problème qui évite de donner trop de connaissances à l'oracle. On suppose que le statut d'un type d'accord au sein d'une

grille peut être précisé par l'accord suivant dans la grille. En effet, certains enchaînements d'accords sont caractéristiques et fixent avec certitude le statut des accords. Ainsi, il est classique pour un improvisateur, en jazz, de s'entraîner à développer son phrasé sur des cadences comme VIm7 IIm7 V7 1M. Par là même, il apprend à repérer dans une grille les enchaînements d'accords comme IIm7 V7, car ils définissent à eux seuls une tonalité imposant une certaine modulation.

Il paraît donc efficace et économique de coder, plutôt que les labels harmoniques relatifs, les transitions harmoniques relatives d'un label à un autre. Ainsi garde-t-on une forte corrélation entre le matériau musical et la contrainte harmonique sous-jacente.

Le codage finalement retenu pour la transition

(degré1 typeAccord1) → (degré2 typeAccord2)

est :

((degré2 – degré1) typeAccord1 typeAccord2)

On parlera dorénavant de *« transition harmonique relative »*.

VII.2.4 Des oracles mélodiques : « l'Orack »

Une fois la structure d'oracle rythmique mise en place, il a fallu trouver un moyen de générer des phrases mélodiques pouvant y être apposées. La première démarche, la plus frustre, a consisté à utiliser des phrases générées par un oracle beat classique, et à tenter de les recoller aux phrases rythmiques. On se soucie du respect des harmonies de la grille d'accords de référence, ce qui nous a contraint à utiliser un oracle dont la fonction de comparaison est celle des labels harmoniques. Néanmoins, il se pose le problème du nombre de notes à générer sur chaque label harmonique, et ce contrôle n'existe pas dans l'oracle beat standard. De plus, on peut prétendre à une granularité mieux adaptée à la manipulation du phrasé mélodique.

C'est ainsi qu'est apparue l'idée d'utiliser des oracles assimilant des jeux de pitchs (comme en mode free), chacun étant dédié à une transition harmonique relative.

Pour faciliter l'utilisation d'oracles multiples, il est paru plus simple de mettre en place une structure spéciale, un « rack d'oracles » dont la classe a été baptisée « Orack ». Les objets de cette classe contiennent trois champs :

_ Rack est un tableau de taille variable dont les éléments sont des oracles. On utilise une fonction dédiée pour créer les oracles au sein de cette structure.

_ NbOracles est le nombre d'oracles contenus dans « Rack ».

_ IndexOracles est un tableau d'étiquettes permettant d'indexer les oracles contenus dans « Rack ». Ce sont des « clefs » d'accès aux oracles.

La méthode **learn-eventlist-orack** permet de réaliser l'apprentissage d'une séquence dans un oracle dont on précise la « clef ». Si la clef est présente dans la liste IndexOracles, l'apprentissage est réalisé avec l'oracle qu'elle étiquette. Sinon, un nouvel oracle est créé pour la nouvelle clef, qui est ajoutée à la liste IndexOracles. Les clefs qu'on utilise sont les transitions harmoniques relatives. Le

type d'objet sur lesquels est réalisé l'apprentissage est fondé sur le modèle des CrossEvents.

L'idée est donc, pour chaque rytbeat sur lequel est réalisé l'apprentissage de l'oracle rythmique, de transformer son midiset en une séquence de CrossEvents par les méthodes de quantification développées par O. Lartillot (cf. [3]). On réalise ensuite l'apprentissage de cette séquence dans l'Orack, en utilisant la transition harmonique relative comme clef. Comme on veut uniquement constituer des dictionnaires mélodiques, on supprime de la séquence les CrossEvents dont le midiset est vide.

On a finalement mis en place un système d'oracles dédiés à chaque transition harmonique relative, à l'intérieur desquels les pitchs sont stockés de façon relative (i.e. transposés comme si le degré du label harmonique de référence était 0, c'est-à-dire do). On établit ainsi des *dictionnaires mélodiques dédiés aux transitions harmoniques relatives*. Pour chaque type de transition harmonique relative, on peut donc générer des phrases mélodiques en utilisant les propriétés de l'oracle, c'est-à-dire des caractéristiques « contexte-inférence » à l'échelle des notes.

VII.2.5 Recollement rythmico-mélodique

Pour des raisons pratiques, on a voulu avoir des objets contenant plus de champs que les CrossEvents. On a créé pour cela la classe CrosseventPlus. Comme en mode free, ces objets correspondent à des tranches polyphoniques. De plus, avant apprentissage, on a supprimé les CrossEventsPlus de midiset vide. Les objets de la classe CrossEventPlus possèdent donc des champs supplémentaires, outre ceux de la classe CrossEvent :

_ Velocite est la somme des vélocités des notes du midiset. On pense pouvoir utiliser ce champ pour repérer des accentuations particulières des phrases mélodiques.

_ Maxduree est un booléen destiné à marquer les CrossEventPlus correspondant à des maxima locaux de durée.

Ces deux champs ont été ajoutés dans le but de retrouver une corrélation entre les parties rythmique et mélodique générées séparément.

En effet, le recollement des phrases mélodiques sur les rythmes générés par l'oracle rythmique s'effectue jusqu'à maintenant de manière totalement dé corrélée. Le protocole actuel, pour générer une improvisation sur la grille de référence consiste en :

_ Générer une improvisation rythmique de la durée d'une grille.

_ Séparer cette improvisation suivant les labels de la grille.

_ Pour chaque label (transition harmonique relative), compter le nombre de notes à générer, et effectuer une génération de cette taille avec l'oracle mélodique approprié (on transpose les pitchs pour qu'ils correspondent au label harmonique). Notons que le nombre de notes à générer pose un problème non trivial, puisqu'on n'a pas de tel contrôle sur l'oracle. On se contente pour l'instant d'en générer trop, en abandonnant ensuite le surplus.

_ Effectuer un « matching » terme à terme entre la séquence rythmique et la séquence mélodique.

On conçoit bien que la perte d'information due à la décorrélation rythme/mélodie se fait sentir : les improvisations générées n'ont pas la même cohérence que celles d'un improvisateur humain. On espère, à terme, retrouver cette cohérence, par exemple en utilisant des marqueurs de « temps fort » ou de « note dans l'harmonie / hors harmonie ».

VII.2.6 Tests

Les tests ont principalement été réalisés à partir d'enregistrements au format midi du pianiste B. Lubat utilisant Omax en mode beat. Grâce à un patch MAX/MSP de simulation, on peut opérer un nouvel apprentissage de l'oracle à partir de ces enregistrements.

Si le fonctionnement d'Omax dans ses modes classiques est particulièrement convaincant, et permet une véritable interaction avec l'instrumentiste (tests réalisés à l'IRCAM avec M. Chemillier), l'oracle rythmico-mélodique n'en est encore qu'à ses débuts. Du point de vue rythmique, la caractérisation des patterns au niveau du beat fonctionne comme on l'espérait : les patterns reconnus sont simples. La quantification rythmique est assez violente, mais c'est nécessaire si l'on veut que l'oracle ait des critères de comparaison efficaces entre les rytbeats.

Par contre, un sérieux problème se pose avec la polyphonie. En effet, l'opération de recollement rythme/mélodie telle qu'on l'a on a décrite est particulièrement frustre, et perd toute notion de polyphonie. Autrement dit, si l'oracle rythmique génère des phrases où sont joués des accords, on risque, lors du recollement, d'« écraser un phrasé monophonique » en un accord. Les traits chromatiques donnent ainsi naissance à des clusters totalement hors de propos dans la caractérisation du style musical de l'instrumentiste. Par contre, le phénomène inverse, d'« éclatement d'un accord en phrase monophonique » est largement moins gênant.

Pour tenter de caractériser le comportement du système, il est judicieux de le tester en utilisant conjointement des improvisations sur différents registres, ainsi que des patterns rythmiques simples et variés.

Finalement, on peut dire que le protocole donne des résultats encourageants lorsqu'on se contente d'utiliser des phrasés monophoniques. On utilise en effet des traits caractéristiques de l'instrumentiste, en les remaniant totalement. On peut déjà supposer que l'« inventivité » de ce système est à même de stimuler grandement les instrumentistes amateurs d'improvisation.

VIII. PROSPECTIVES

Le travail présenté dans ce rapport est, par bien des côtés, inachevé. Il se contente d'ouvrir des portes, encourageant en cela de futurs développements du système Omax. La démarche qui y a présidé est fortement exploratoire, puisqu'on a cherché à mettre en place des solutions efficaces et assez légères, dans le but d'avoir à disposition les outils permettant de tester de nouvelles extensions des capacités du système.

Cette partie est dédiée aux prospectives, idées paraissant raisonnablement réalisables à partir de l'architecture que l'on a décrite tout le long de ce document.

Utilisation de l'Oracle

Si l'oracle des facteurs est particulièrement convaincant dans l'utilisation qu'on en fait, ainsi que pour les raisons théoriques développées plus haut, il n'en reste pas moins que son utilisation pour la génération de séquences n'est que très lâchement contrôlée. On ne peut actuellement valider le protocoles de navigation/génération au sein du graphe qu'à l'écoute des improvisations synthétisées.

Aussi serait-il particulièrement avantageux de mener une étude statistique précise des parcours du graphe effectués à la génération, afin d'améliorer les algorithmes utilisés. En particulier, on peut se demander sous quelles conditions le parcours de l'oracle crée des cycles (une séquence apparaissant périodiquement, toujours au même moment de la grille harmonique) ou tombe dans des puits de répétition (bégaiements dans les improvisations générées).

Oracles parallèles

La mise en place des oracles parallèles de l'Orack permet d'en envisager de nouveaux usages. Sur un morceau donné, on peut par exemple imaginer réaliser l'apprentissage des oracles à partir de différents musiciens. Chaque oracle donnerait alors accès à un style d'improvisation différent. Il vient alors à l'idée de créer un protocole d'hybridation des styles d'improvisation, en mélangeant les improvisations générées par différents oracles.

Dans le même ordre d'idées, G. Assayag et M. Chemillier travaillent actuellement à la mise en place d'un « orchestre d'oracles », utilisant une structure d'oracles multiples dédiés (à terme) à différents instrumentistes midi.

De plus, les système de transition harmoniques relatives ayant été mis en place, il devient envisageable, moyennant un apprentissage assez conséquent, de mettre sur pied un oracle « tout terrains », capable d'improviser sur n'importe quelle grille d'accords. Ceci est, en l'état actuel des choses, déjà réalisable, mais il faut prendre le temps de réaliser cet apprentissage.

Protocole de communication MAX / Open Music

Notons que le protocole d'échange de données entre MAX/MSP et Open Music n'est pas particulièrement satisfaisant. En effet, on utilise Midishare, un protocole de transmission de flux midi, pour transmettre des données d'un autre format (grille d'accords par exemple), ou encore pour effectuer des requêtes (appels à l'exécution de fonctions).

On envisage, dans le but de « nettoyer » le système, d'utiliser Open Sound Control (OSC), a priori plus généraliste. Mais son implémentation dans Open Music n'est pas réalisée, et peut-être est-il plus sage d'attendre le portage définitif d'Open Music sous OSX avant de franchir cette étape.

On pourrait, grâce à OSC, envisager une extension d'Omax à un système distribué, avec plusieurs machines. Dans un cas comme celui de l'« orchestre d'Oracles », cela permettrait d'utiliser des machines dédiées afin de multiplier la puissance de calcul disponible.

Recombinaisons rythmico-mélodiques

En ce qui concerne l'oracle rythmico-mélodique, on a évoqué plus haut les problèmes de recollement rythme/mélodie. Il est évident que le travail n'est, de ce point de vue, que commencé. Il s'agit de trouver les marqueurs et méthodes adéquats pour recréer la corrélation rythme/mélodie qui existe chez un improvisateur humain. Des pistes de travail ont été mises en place avec les objets CrossEventPlus (marqueurs de temps forts, de début de phrase mélodique, maxima et minima locaux de pitch ou de vélocité), mais il reste à en imaginer les possibilités réelles.

Une telle approche n'est peut-être pas viable sans l'ajout de connaissances musicales, ce qui n'était, aux débuts d'Omax, pas le propos.

Mode free-beat

Sur une idée de G. Bloch, on peut imaginer une utilisation de l'oracle rythmique dans un mode d'interaction rythmiquement moins contraint que le mode beat. L'idée serait de constituer un véritable mode free-beat dans lequel un instrumentiste improvise totalement librement, et sans accompagnement. A partir de son improvisation, il s'agirait d'*inférer le tempo* de courtes séquences pour procéder à une analyse rythmique de type rytbeat telle qu'on l'a décrite. Si en plus on stocke le phrasé rythmique de façon *relative*, il est possible de générer des improvisations à n'importe quel tempo. Une analyse statistique du tempo de l'instrumentiste permettrait de recréer les profils d'évolution du tempo, on irait ainsi encore plus loin dans l'imitation du style, avec des structures de plus haut niveau qu'actuellement.

Dans le même ordre d’idées, on peut imaginer, en mode free, inférer des labels harmoniques à partir de la partie basse du clavier, dans le but d’analyser les constructions harmoniques à un niveau plus symbolique.

IX. APPENDICE (CODE LISP)

Définitions des classes d'objets

```
(defclass* orack ()
(
(rack :initform (make-array 10 :adjustable t) :accessor rack :initarg :rack)
(nboracles :initform 0 :accessor nboracles :initarg :nboracles)
(indexOracles :initform () :accessor indexOracles :initarg :indexOracles  ))

(defclass* event ()
(
(MidiSet :initform () :initarg :MidiSet :accessor MidiSet)
(duration :initform 500 :initarg :duration :accessor duration)
))

(defclass* beat (event)
(
(HarmLabel :initform () :initarg :HarmLabel :accessor HarmLabel)
(RelHarmLabel :initform () :initarg :RelHarmLabel :accessor RelHarmLabel)
(NumBeat :initform 1 :initarg :NumBeat :accessor NumBeat) ; in the measure
(RefHarmLabel :initform 1 :initarg :RefHarmLabel :accessor RefHarmLabel)
(StartPhrase :initform () :initarg :StartPhrase :accessor StartPhrase)
(QMidiSet :initform () :initarg :QMidiSet :accessor QMidiSet)
(Qdivision :initform 1 :initarg :Qdivision :accessor Qdivision)
(Density :initform 1 :initarg :Density :accessor Density)
))

(defclass* rytbeat (beat)
 (NextHarmLabel :initform () :initarg :NextHarmLabel :accessor NextHarmLabel)
(HarmTrans :initform () :initarg :HarmTrans :accessor HarmTrans)
(Pitchtree :initform () :initarg :Pitchtree :accessor Pitchtree)
(Rytree :initform () :initarg :Rytree :accessor Rytree)
(Ratio :initform () :initarg :Ratio :accessor Ratio)
(Duration :initform 0 :initarg :Duration :accessor Duration)
(Eventlist :initform () :initarg :Eventlist :accessor Eventlist)
(Nbnotes :initform () :initarg :Nbnotes :accessor Nbnotes)
(PreBeatMS :initform () :initarg :PreBeatMS :accessor PreBeatMS)
```

```
(PostBeatMS :initform () :initarg :PostBeatMS :accessor PostBeatMS)
))

(defclass* CrossEventPlus (event)
(
(Velocite :initform 0 :accessor velocite :initarg :velocite)
(MaxDuree :initform 0 :accessor maxduree :initarg :maxduree).
(StartBeat :initform 0 :accessor startbeat :initarg :startbeat)
))
```

X.BIBLIOGRAPHIE

[1] C. Allauzen, M. Crochemore, M. Raffinot. Oracle des facteurs, oracle des suffixes. *Rapport technique de l'institut Gaspard Monge*, 1999.

[2] A. Lefebvre, T. Lecroq. Computing repeated factors with a factor oracle. 2000.

[3] O. Lartillot. Modélisation du style musical par apprentissage statistique : une application de la théorie de l'information à la musique. *Rapport de stage DEA ATIAM*, 2000.

[4] E. Poirson. Simulation d'improvisation à l'aide d'un automate de facteurs et validation expérimentale. *Rapport de stage DEA ATIAM*, 2002.

[5] O. Lartillot, S. Dubnov, G. Assayag, G. Bejerano. Automatic modeling of musical style. *Proceedings of Journées de l'informatique musicale*, 2001.

[6] G. Assayag, Dubnov, Delerue. Guessing the composer's mind : applying universal prediction to musical style. *Proceedings of the International Computer Music Conference*, 1999.

[7] M. Steedman. The blues and the abstract truth, 1999.

[8] R. Rowe. Systèmes musicaux interactifs. *The MIT Press*, 1993.

[9] R. Rowe. Machine musicianship. *The MIT Press*, 2001.

[10] J. Langner, W. Goebl. Visualizing expressive performance in tempo-loudness space. *ESCOM 10th anniversary conference on musical creativity, Liège, Belgium*. 2002.

[11] S. Dixon, W. Goebl, G. Widmer. The performance worm : Real time visualisation of expression based on Langner's tempo-loudness animation.

[12] Exemple LZ77 par Hassin.
http://www.epita.fr.8000/hassin_a/ppp/lz77/lz77.html. *Internet*,1998.

[13] D. M. Franz. Markov chains as tools for jazz improvisation analysis. *Master of science Thesis*, 1998.

[14] N. RUWET. Méthodes d'analyse en musicologie. *Revue Belge de musicologie*, (20) :65-90,1966.

[15] D. COPE. Signatures and earmarks : Computer recognition of patterns in music. *Computing in musicology 11 (1997-98), pp. 129-138. The MIT Press.*