# Simulation d'improvisations à l'aide d'un automate de facteurs et validation expérimentale

Mémoire de stage de DEA ATIAM

Emilie Poirson

Equipe Représentations Musicales IRCAM CNRS UMR 9912
LEAD Université de Bourgogne CNRS UMR 5022

Responsables du stage : Gérard Assayag et Emmanuel Bigand

1er juillet 2002

Université Pierre et Marie Curie (Paris VI)

# Table des matières

# Table des figures

# I Introduction

Un des objectifs de l'IRCAM (Institut de Recherche et de Coordination Acoustique et Musique) est d'apporter aux compositeurs d'aujourd'hui les outils et matériaux nécessaires à la création dans un environnement sans limite. C'est ce à quoi travaille l'équipe " Représentations Musicales ", dirigée par Gérard Assayag.

Dans ce domaine, apparaît très vite l'ambivalence de la place de la machine par rapport à l'homme. L'ordinateur peut-il remplacer le musicien dans ce qui le caractérise le plus : la création ? Pour cela, il semble évident que la machine passe, comme ce fut le cas pour l'homme, et parfois de manière implicite, par une phase d'apprentissage. La problématique d'introduire et d'organiser les données une fois résolue, il faut ensuite concevoir un processus de création. L'algorithme se substitue alors à l'agencement et à l'exploitation de l'expérience acquise par le virtuose pour produire de la musique.

L'étude qui suit se rallie à cet ambitieux projet. Elle s'articule autour de trois phases :

- modéliser un style musical produit par un musicien,
- simuler une improvisation à partir de ce style,
- en mesurer la performance.

Son périmètre est restreint au domaine du jazz, en tant que style improvisé prédominant de notre époque.

Pour modéliser le style, il faut tout d'abord en avoir une bonne définition et ainsi appréhender les paramètres nécessaires à sa représentation. La première partie définit les notions de base de notre sujet. Dans un second temps, nous envisageons différentes méthodes d'analyse, puis les évaluons, pour en dégager une seule, qui répond au mieux aux objectifs et contraintes de la simulation.
La dernière partie est consacrée à l'étude psycho-acoustique des d'échantillons produits. Elle présente la mise en œuvre et les résultats d'expériences réalisées afin de valider notre travail.

# II Exposition du sujet en détail

## II.1 Style musical et jazz

Il semble qu'il n'y ait pas de définition empirique du style, et c'est ce qui le rend très dur à modéliser. Pour Rosen [28], « on pourrait, en langage figuré, appeler style une manière d'exploiter et de faire converger les éléments d'un langage, cette manière constituant ensuite un même « dialecte » ou un « langage » ; cette convergence définit ce qu'on peut appeler style ou manière de l'artiste ». Ainsi, dans le même contexte, chaque artiste a sa propre façon de combiner les symboles élémentaires ; il se crée ses propres règles, qui lui donnent son identité. Chacun a son langage : même si les éléments de base peuvent être les mêmes, la façon de les utiliser, de les organiser est personnelle et difficile à décrire. La preuve est que, le plus souvent, on ne sait pas décrire un style. On se contente de le comparer à d'autres. D'ailleurs, les compositeurs "s'amusent" à faire des pièces ou des variations "dans le style de" tel ou tel compositeur. Il est plus facile de trouver des points communs avec d'autres styles connus que les mots justes. Du reste, les ressemblances ne sont pas fortuites : un style est créé par des choix d'inspirations, de références du passé. On prend des éléments chez plusieurs artistes que l'on a entendu, étudié et on les organise à sa façon, pour créer quelque chose de nouveau, inspiré mais pas dans le style d'un autre.

On parle souvent de la différence de style entre les pianistes comme Keith Jarrett, Herbie Hancock, Duke Ellington et bien d'autres. Et ces styles individuels appartiennent à un style de groupe, d'époque : le jazz. Le jazz, né dans le sud des Etats-Unis d'Amérique au début du 20ème siècle, est une musique où l'on retrouve des éléments de la musique européenne et de la musique africaine. Ces deux cultures se sont mélangées pour donner cette manière de jouer qui semble si naturelle. Ces deux cultures mélangées, mais aussi, le paradoxe énorme qu'il existe entre elles. Alors, pour que chacun s'exprime malgré les différences, c'est sur l'improvisation que le jazz se base et c'est cette forme d'improvisation que nous étudions.

## II.2 Qu'est-ce que l'improvisation ?

Pour Jean-Pierre Leguay, « Improviser, individuellement ou collectivement, consiste à concevoir et réaliser, dans l'instant, tout ou partie d'un projet musical, à inventer et à jouer spontanément *sa* propre musique ». L'improvisation permet donc de faire fi des contraintes, des données arbitrairement imposées. Elle permet le bavardage à ceux qui en ont besoin, alors que la partition oblige le chemin, ne laisse pas le choix.

On qualifie donc d'improvisée une musique non écrite, comprenant un fort degré d'indétermination. Le champs est laissé libre à l'interprète qui revêt alors l'habit de compositeur et choisit un chemin personnel pour exprimer ses idées. C'est pourquoi l'improvisation est souvent synonyme de liberté. Chacun fait ses règles, organise ses idées sur le vif. L'improvisation est une sorte de composition instantanée. « Le musicien improvisateur procède alors instantanément à la traduction fidèle ou filtrée de ses rêves en figures sonores ». Ainsi, tout comme l'interprétation d'un morceau, une improvisation n'est pas reproductible puisqu'elle correspond à un état d'esprit, à un instant. Mais la différence est dans le fait que l'improvisation est inouïe et surtout éphémère alors que la pièce écrite est faite pour durer. Ce sont deux modes de *composition* très différents et dont les partisans ne s'accordent pas forcément. Le temps et la patience nécessaires à un compositeur le rend indifférent à un mode de jeu où l'on doit créer dans l'instant et inversement. Pas question de revenir en arrière, de se reprendre, d'effacer une note ou de modifier une mélodie. Pas question de penser qu'à ce moment-là, les gens n'ont peut-être pas envie d'entendre ça et de réfléchir à ce qui serait plus approprié. « L'improvisateur joue la musique que, présentement, il désire entendre. Il joue immédiatement une part de son monde sonore qu'il vit mentalement, en dépendance directe de ce que son expérience directe et sa culture lui permettent d'en capter et d'en traduire instrumentalement ».

L'improvisation est donc très personnelle mais pas égoïste. Il n'est pas rare que les musiciens improvisent à plusieurs , ou soient accompagnés. Alors, ils doivent instaurer un dialogue, construire ensemble une histoire. Le monde de l'improvisateur est alors largement influencé par ce que propose les autres musiciens. Elle nécessite des réflexes, des réponses instinctives, impulsives, une assise pour pouvoir prendre des décisions rapidement, la faculté de répondre justement. Mais tout cela n'est pas inné. Si certains sont plus à l'aise au départ pour s'exprimer de la sorte, chacun utilise des objets de son bagage musical, de son propre dictionnaire. Mais, ce dictionnaire est bien formé par une phase d'apprentissage.

Pour l'American Heritage Dictionary, c'est « the act of inventing, composing, or reciting without preparation ». Cette dernière expression, *sans préparation* est capitale. En effet, comme dit précédement, l'improvisation est de l'organisation d'idées dans l'instant donc, prendre les décisions en "temps réel", sans y avoir pensé auparavant. Cependant, sans préparation ne signifie pas que l'on ne part de rien pour une improvisation. Elle doit être précédée d'une période d'apprentissage par l'écoute, la connaissance du style, de l'instrument. Si l'interprète doit organiser ses idées, il doit pouvoir les for-

muler, organiser des riffs, des boucles, des séquences travaillées ou entendues auparavant qui sont venues enrichir le langage musical personnel. Ainsi on peut reconnaître un instrumentiste par son style d'improvisation, puisque chacun utilise ses outils propres. Dans une improvisation, on peut retrouver la culture, l'éducation, le passé du musicien. C'est pourquoi un musicien classique ayant le même bagage technique qu'un musicien de jazz ne sera pas à l'aise dans cet exercice car il n'aura pas les références. A l'inverse, le musicien de jazz se sentira bridé devant une partition. C'est un apprentissage différent. L'improvisation semble ouverte, libre, imprévue, contrairement à une pièce écrite entièrement.

L'improvisation est donc une composition sur le vif, en prenant composition à son sens premier : la formation par assemblage de plusieurs éléments. Ces éléments ont été appris, consciemment ou non et sont différents selon chaque musicien qui forme avec cela un discours cohérent.

## II.3 L'apprentissage implicite

*Apprentissage : méthode permettant d'établir des connexions entre certains stimuli et certaines réponses, dont le résultat est d'augmenter l'adaptation de l'être vivant à son milieu.*

On peut définir l'apprentissage comme le rangement en mémoire d'informations, pour les réutiliser ultérieurement.

*Implicite : qui est contenu dans une proposition sans être exprimé en termes précis, formels.*

L'apprentissage implicite se définit donc comme la prise en compte d'information, de manière inconsciente. Reprenons les premières lignes de Meulemans [18] : « La notion d'"apprentissage implicite" renvoie à notre capacité d'apprendre, sans que nous en soyons conscients, des informations de nature complexe, et au fait que la connaissance acquise est elle-même difficilement accessible à la conscience (ou, en tous cas, à l'évocation verbale) ». Le sujet se rend compte qu'il a appris quelque chose mais est incapable de l'expliciter, de le formuler avec des mots. Cette intuition de nouvelles connaissances semble logique. Tout le monde a appris à marcher, à parler...mais comment ? Personne ne se souvient d'avoir appris à apprendre. Pour résumer cette définition, « l'apprentissage se fait à l'insu du sujet, et la connaissance acquise est difficilement accessible à la conscience ou exprimable verbalement ». On a donc l'impression d'avoir appris quelque chose mais on ne peut

pas décrire ce que l'on a appris.

Les recherches en neuropsychologie sont importantes pour tenter d'expliquer le mécanisme d'apprentissage implicite. Mais, elles sont surtout très difficiles à mener puisqu'il s'agit de faire une relation entre les aptitudes cognitives et ce mécanisme d'apprentissage qui, par définition, n'est pas descriptible par le sujet. Il n'y a donc pas de références, donc, pas de description des processus neuronaux, d'où beaucoup de questions sur les régions du cerveau entrant en jeu. Une étude qui peut être cependant menée est une comparaison entre un sujet dit normal et un sujet dans un état amnésique. Comme dit précédemment, apprendre, c'est ranger des informations en mémoire pour les réutiliser plus tard. Ce que l'on appelle amnésie est un "formatage" de cette mémoire. Or, il se trouve que l'apprentissage implicite est préservé dans ces conditions : la parole est retrouvée alors que l'identité même est oubliée. Il est heureux que cela soit ainsi car, si une personne perdait tout ce qu'elle a appris, elle perdrait en même temps cette faculté d'apprendre. Cela prouve qu'il y a donc bien deux mécanismes différents : l'un conscient, explicite, et l'autre inconscient, implicite. Pour Reber, le second serait un processus précoce, à la base des premiers développements de l'enfant. Il serait activé avant même que l'enfant se mette à parler, et ne faiblirait pas avec les années. « Children must be capable of implicit acquisition of complex knowledge of their environment because that's the way they do ». D'après lui, les structures neuronales impliquées se développeraient plus vite que celles de la mémoire déclarative. Cela expliquerait que les bébés comprennent les choses rapidement : un enfant apprend implicitement à reconnaitre ceux qui s'occupent de lui et si un problème survient, il sent le changement dans son entourage.

L'adaptation au milieu, les changements des représentations conscientes pour une meilleure correspondance avec les unités de l'environnement ne se font pas consciemment. Pour Perruchet, « l'apprentissage implicite façonne notre expérience phénoménologique du monde ». Une personne va adapter son comportement à la situation à laquelle elle est exposée mais cette adaptation n'est pas volontaire, intentionnelle. On peut faire ici le lien avec l'improvisation en jazz : le musicien s'adapte au milieu, au morceau, aux autres musiciens. Son improvisation vient bien de lui, mais il ne peut expliquer pourquoi il a choisit cela et il ne savait peut-être même pas qu'il connaissait cela. On dit que plus on joue, plus on se découvre : plus on découvre les choses apprises implicitement.

Le style, l'improvisation, l'apprentissage implicite : ces trois notions sont fondamentales dans un tel sujet. Nous cherchons donc à savoir, dans un pre-

mier temps, s'il est possible de modéliser un style improvisé, en restreignant le domaine au jazz, et dans un second temps, si une simulation informatique peut faire illusion face à un musicien, ce dans le but de les faire jouer ensemble par la suite. Pour modéliser ce style, nous allons étudier quelques algorithmes, tous basés sur les arbres de recherche que nous définirons dans une première partie.

# III Représentations informatiques

Notre projet est d'évaluer une série d'algorithmes qui permettent d'analyser des séquences musicales, d'en déduire un modèle, puis de générer des improvisations à partir de ce modèle.
Ces modèles seront toujours des modèles contexte-continuation, c'est-à-dire que l'on cherche à repérer dans le texte musical des contextes (patterns significatifs par leur redondance) et des continuations (les symboles musicaux par lesquels ces patterns ont le plus de chance de poursuivre).
Ces modèles ont un lien très fort avec les chaînes de Markov mais représentent un progrès par rapport aux applications connues de ces chaînes en musique, dans la mesure où les contextes ne sont pas de longueur fixe, mais au contraire, s'adaptent en fonction de la morphologie de la séquence analysée. En ce sens, on pourrait parler de modèle markovien adaptatif.
Dans cette recherche, on privilégiera les approches donnant lieu à des algorithmes incrémentaux, linéaires en temps et en espace, de manière à pouvoir aborder plus tard une application temps réel.

## III.1 Les modélisations

L'automate semble être la structure la mieux adaptée pour faire de la génération car, chaque état indique la façon dont il faut continuer.
D'un autre côté, pour faire de la génération, on fait de la recherche de patterns, qui est très simple avec une structure d'arbre de suffixe.
Il serait intéressant pour nous d'avoir un modèle qui combine ces deux avantages.

Dans la plupart des représentations que l'on va étudier, on aura une structure d'arbre représentant la logique d'organisation des patterns, et d'autre part, ces arbres seront utilisés comme des automates.

Dans le dernier modèle étudié, et celui retenu (défini en IV.1.1), il ne s'agira pas d'un arbre mais d'un automate de facteurs. Cependant, nous verrons que celui-ci se déduit d'un arbre de suffixe.

### III.1.1 Les arbres

Un arbre est une modélisation, basée sur une grammaire et pouvant être représenté graphiquement. Ce chapitre s'attache à expliquer de manière plus approfondie quelques notions sur les arbres nécessaires pour la suite de l'étude.

1. Structures et opérations

   Comme dit précédemment, l'arbre est utilisé ici comme un automate. On définit la structure des automates par un état source et des états finaux, avec des nœuds de transition.
   Les automates considérés ont un graphe en forme d'arbre. C'est donc également ainsi que l'on parlera de l'arbre.
   Les flèches sont étiquetées par les éléments de l'alphabet. Ainsi, pour la génération, il suffit de parcourir l'arbre et de relever la continuation.
   Il existe différents types d'arbres dont les arbres non compactés.
   Voici pour exemple deux opérations sur les arbres, simplifiant leur représentation.

   – *Compactage*

     On remarque que parfois, il y a de longues branches sans bifurcation (choix).
     Par exemple :

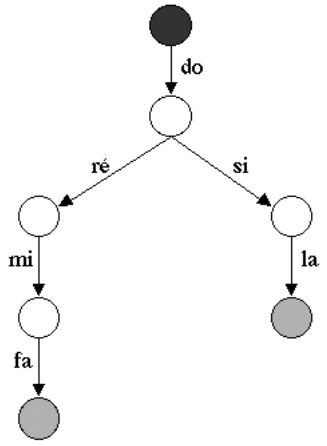


FIG. 1 – Arbre non compacté.

En noir, l'état initial. En gris les états finaux. On voit que , une fois le choix do-ré ou do-si fait, on va jusqu'à l'état final sans avoir à faire de choix. On peut donc "compacter" les dernières branches, comme le montre la figure 2 page 12.

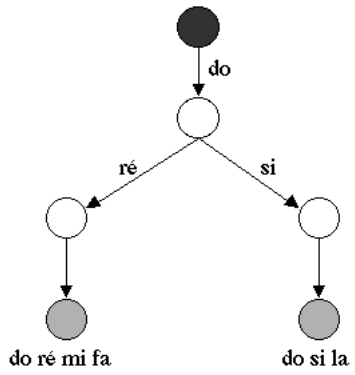


FIG. 2 – Arbre compacté.

L'état terminal est atteint avec do-ré-mi-fa ou do-si-la. Si l'on choisit d'aller vers le ré, on sait déjà que notre mot sera do-ré-mi-fa en état final.

– *L'opération "collapse"*
Pour simplifier un arbre, on peut également réduire les branches sans choix de cette façon :

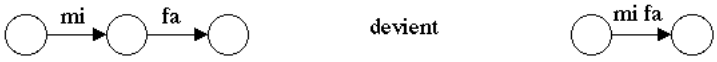


FIG. 3 – L'opération "collapse"

Maintenant, au lieu de 3 nœuds, 2 suffisent. Et comme, généralement, les arbres ont énormément de noeuds, ces simplifications soulagent l'éxécution des analyses.

### III.1.2 Les tries

1. Définition
Les "tries" sont des structures de données qui permettent trois opérations fondamentales :
   - Ajout (INSERT)
   - Suppression (DELETE)
   - Recherche (FIND)

Le résultat de cette dernière opération peut être soit le résultat du test booléen d'appartenance à l'arbre, soit la valeur associée à l'argument.

2. Suffix tries et suffix trees
On appelle "suffix trie" l'arbre formé par les caractères d'un mot et de tous ses suffixes. Avec les premières notes du fameux 1er prélude de Bach, on doit retouver dans l'arbre :

Do3 Mi3 Sol3 Do4 Mi4 Sol3 Do4 Mi4 $
Mi3 Sol3 Do4 Mi4 Sol3 Do4 Mi4 $
Sol3 Do4 Mi4 Sol3 Do4 Mi4 $
Do4 Mi4 Sol3 Do4 Mi4 $
Mi4 Sol3 Do4 Mi4 $
Sol3 Do4 Mi4 $
Do4 Mi4 $
Mi4 $

FIG. 4 – Ensemble des suffixes de la mélodie de Bach

Un tel arbre peut être transformé en arbre de suffixes (suffix tree) en le compactant et en le "collapsant" . Ce qui donnerait :

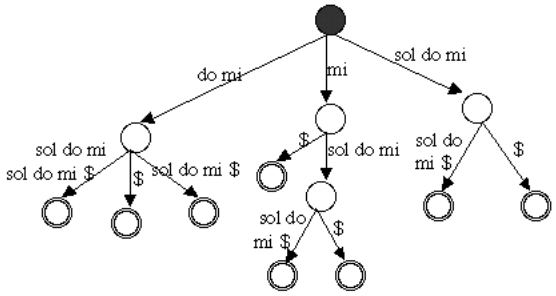


FIG. 5 – Suffix tree de la mélodie de Bach.

On peut ensuite rechercher des mots dans l'arbre en partant de la racine et en suivant les flèches et les continuations possibles ou non. Si l'on se trouve coincé dans l'arbre, c'est que le mot n'appartient pas au texte initial. Notons également que pour un mot appartenant à l'arbre, il n'y a qu'un seul chemin d'accès. Si l'on recherche un mot, on ne peut pas prendre « un mauvais chemin ». C'est un avantage des arbres de recherche : tester l'appartenance d'un mot est simple et rapide.

Pour construire ces arbres de recherche, il existe différentes méthodes. Nous en avons choisi 3 parmi toutes celles existante. Notre choix s'est fait en gardant toujours comme objectif ce projet temps réel. Nous nous sommes donc orientés vers des algorithmes incrémentaux.

## III.2 Différents algorithmes

Lempel-Ziv, PST et Ukkonen sont les trois algorithmes de construction d'arbres que nous allons expliciter dans ce paragraphe. Nous en décrirons le fonctionnement, les avantages, les inconvénients, ce qui nous a poussé à ne pas les exploiter par la suite.

### III.2.1 Lempel-Ziv

1. LZ
   LZ est un algorithme de compression de données développé par Jacob Ziv et Abraham Lempel [21],[19]. L'avantage de cette méthode est qu'elle peut s'adapter à tous les supports : textes, images et sons. Il existe deux sortes d'algorithme LZ . Tout d'abord, et nous n'en donnerons qu'une brève description, les algorithmes à fenêtre coulissante LZ77. Ensuite, et ce sont ceux qui nous intéressent le plus, les algorithmes à base de dictionnaire.
   - *LZ77*
     Cette méthode 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 dictionnaire (donnant début et taille de la chaine).

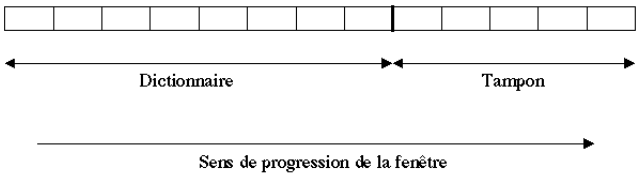


FIG. 6 – Méthode LZ77

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

Les avantages :
Cet algorithme permet de ne pas coder les redondances, souvent présentes dans un texte avec les "le", "de", "... sion", "... ment"...

Cela soulage la quantité d'information donc, compresse les données. Autre avantage : le codage est simple et très économe dans le cas d'un motif répété plusieurs fois à suivre. Notons que la décompression est simple et rapide puisqu'il suffit de "suivre" les pointeurs.

Les inconvénients :
Malgré un principe simple, sa programmation nécessite une gestion de pointeurs glissants, de tableaux à longueur variable d'objets à longueur variable... Elle en devient complexe. De plus, à chaque pas, on compare l'élément du tampon à tous ceux du dictionnaire. Au début, peu de mots donc, comparaison rapide mais dès que le dictionnaire s'alourdit, le temps de compression augmente avec le nombre de comparaisons. Donc, dans des données longues, l'algorithme n'est pas efficace. On a montré au-dessus l'efficacité quand au traitement des répétitions. Certes, dans un texte, on trouve souvent les mots de liaison, les articles et même les mots-clés du texte. Mais lorsque le support ne contient pas (ou peu) de répétition, on code tous les éléments avec 3 champs (position, longueur, suivant), ce qui fait trois fois plus d'informations en mémoire donc, exactement l'inverse d'une compression !!
Pour conclure, LZ77 est à efficacité variable selon le support sur lequel il est utilisé. On peut retrouver quelques exemples en [23].

Les algorithmes à base de dictionnaire

- *LZ78*
  Notons que LZ a fait l'objet d'un sujet de stage de DEA par Olivier Lartillot[20], qui a en programmé une librairie sous Open Music. Le sujet de cette recherche portait sur la modélisation du style. Le principe est le même que notre étude : dans la librairie LZ, des méthodes analysent le morceau et d'autres vont ensuite générer un nouvel extrait. Les résultats ne nous semblent pas très convaincants. C'est pourquoi nous sommes revenus sur cette méthode, pour en mesurer les limites et penser aux améliorations.
  Au début, LZ78, que l'on appelera maintenant LZ, part avec un dictionnaire vide. A chaque pas, on cherche la chaîne inconnue la plus longue. L'analyse se fait donc caractère par caractère et sectionne dès qu'une séquence inconnue est trouvée. Au début, on aura donc des chaines très courtes, dont certaines composées d'un unique élément. L'exemple de "ABABABABA" illustre ce codage. Ici, les lettres soulignées sont celles étudiées. Elles disparaissent quand on les met dans le dictionnaire.

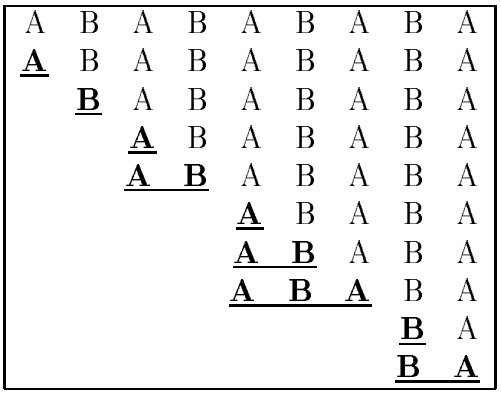


FIG. 7 – Codage du mot ABABABAB

Dans le tableau suivant, on peut suivre le déroulement de l'analyse du mot ABABABAB par LZ. Dans la première colonne, le dictionnaire, vide au départ. Dans la deuxième, l'élément qui se présente à l'analyse et la troisième, l'opération réalisée. Par exemple, à l'étape 1, l'élément A se présente. Il n'est pas dans le dictionnaire, alors on l'y place. Lettre suivante : B. Même opération. Ensuite, encore A. Cette fois-ci, il appartient déjà au dictionnaire, donc, je regarde plus loin : AB est inconnu donc ajouté au dictionnaire. Et cela jusqu'à la fin du mot.

| **Dictionnaire** | **Elt** | **Opération** |
|---|---|---|
| Vide | A | Inconnu : ajout au dictionnaire |
| A | B | Inconnu : ajout au dictionnaire |
| A, B | A | A connu : je regarde le suivant |
| | AB | AB inconnu : ajout au dictionnaire |
| A, B, AB | A | A connu : je regarde le suivant |
| | AB | AB connu : je regarde le suivant |
| | ABA | ABA inconnu : ajout au dictionnaire |
| A, B, AB, ABA | B | B connu : je regarde le suivant |
| | BA | BA inconnu : ajout au dictionnaire |
| A, B, AB, ABA, BA | ... | ... |

FIG. 8 – Codage LZ de ABABABAB étape par étape

On ne code donc d'aucune manière les redondances.
On note que chaque séquence est composée d'un préfixe et d'un suffixe. Le suffixe est le dernier caractère et le préfixe est la chaine privée du suffixe. Par exemple, la chaine ABA : AB est le préfixe et A le suffixe. Pour l'élément A, le préfixe est " " et le suffixe A.
Dans un texte, on peut trouver AB qui est suivi de A ou de B. Alors,

on a ABA et ABB dans le dictionnaire, on dirait que le préfixe AB a deux continuations possibles : A et B.
On peut alors représenter le dictionnaire sous forme d'arbre. Pour notre exemple précedent (en ajoutant ABB au dictionnaire), cela donnerait :

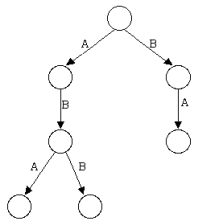


FIG. 9 – Arbre du dictionnaire {A,B,AB,ABA,ABB,BA}

Tous les états sont ici finaux, c'est-à-dire qu'on arrive à tous les états par un mot du dictionnaire.

Les avantages
LZ est un algorithme performant. Les taux de compression ne sont pas négligeables. De plus, il est nettement moins difficile à programmer que LZ77 décrit précédement. Une autre différence avec son prédécesseur est qu'au lieu de comparer le mot à ceux contenus dans le tampon, donc, 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.

Les inconvénients
Le choix du dictionnaire dépend de la mémoire de la machine.
Sur des données courtes, LZ n'est pas efficace. En effet, plus l'extrait est court, moins l'arbre qui le représente sera développé. Donc, au moment de la génération, les continuations n'étant pas nombreuses, on risque de retomber toujours sur les mêmes patterns et de tourner en boucle. De plus, on ne s'éloignera pas beaucoup de l'original. Plus l'arbre est développé, plus il y a de chances de créer un enchaînement nouveau.

2. LZW
Lempel-Ziv-Welch

– *Définition*
C'est une reprise de l'algorithme Lempel-Ziv par Terry Welch en

1984. Ce mode de compression est utilisé entre autres, dans le codage des GIF et TIFF. C.f. [24]

A la différence de Huffman [8], autre méthode de compression non étudiée ici, la compression LZW n'a aucun besoin de construire une table de conversion à l'avance. La table se construit au fur et à mesure que la compression se déroule. Dans le cas d'un texte compressé en LZW, chaque nouvelle occurrence d'un mot est remplacée par un pointeur vers la première occurrence rencontrée. Les ratios de compression obtenus avec cette méthode varient entre 1 :1 et 3 :1. Cela peut aller jusqu'à 10 :1. Une démo de son fonctionnement en [17].
Le principe est presque le même que LZ à la différence près qu'au lieu de tout le temps reprendre au caractère suivant le mot que l'on vient de mettre dans le dictionnaire, on repart un caractère avant. De plus, on ne part pas avec un dictionnaire vide mais contenant déjà les caractères simples comme les lettres (majuscules et minuscules) , les caractères spéciaux connus et autres symboles.

– *Exemples*
Prenons l'exemple donné :
/ W E D / W E / W E E / W E B / W E T

| **LZ** | | | **LZW** | | |
|---|---|---|---|---|---|
| **Entrée** | **Connu** | **Dictionnaire** | **Entrée** | **Connu** | **Dictionnaire** |
| / | Non | / | / | Oui | |
| W | Non | W | /W | Non | /W |
| E | Non | E | WE | Non | WE |
| D | Non | D | ED | Non | ED |
| / | Oui | | D/ | Non | D/ |
| /W | Non | /W | /W | Oui | |
| E | Oui | | /WE | Non | /WE |
| E/ | Non | E/ | E/ | Non | E/ |
| W | Oui | | /W | Oui | |
| WE | Non | WE | /WE | Oui | |
| E | Oui | | /WEE | Non | /WEE |
| E/ | Oui | | E/ | Oui | |
| E/W | Non | E/W | E/W | Non | E/W |
| E | Oui | | WE | Oui | |
| EB | Non | EB | WEB | Non | WEB |
| / | Oui | | B/ | Non | B/ |

FIG. 10 – Comparaison des méthodes LZ et LZW

On peut alors comparer les dictionnaires d'analyse de ces deux méthodes :
**LZ** : /, W, E, D, /W, E/, WE, E/W, /WE, T, EB
**LZW** : /, W, E, D , /W, E/, WE, E/W, /WE, ED, D/, /WEE, WEB, B/, /WET

On se rend bien compte du fait que le répertoire est plus développé dans le cas de LZW donc pourrait donner plus de possibilités à la reconstruction. L'automate a plus d'états et surtout, moins d'états puits ou isolés. Cela est intéressant pour le parcours de l'arbre. En effet, si l'ordinateur parcourt l'arbre et arrive à un état puits, il va répéter la même chose jusqu'à ce que quelqu'un lui demande d'arrêter. Il en est de même pour les états isolés. Plus il y en a, plus il va y accéder facilement et comme à chaque fois qu'il ne peut rien faire, il doit repartir du début, on risque d'entendre souvent la même chose. C'est donc un réel avantage pour cet algorithme que de limiter le nombre d'états puits ou isolés.
Remarquons tout de même que EB n'apparaît que dans le dictionnaire LZ. Dans LZW, il n'est présent que après W.

On pourrait penser que les avantages de LZW viennent du dictionnaire de départ non vide, contenant les caractères unité : testons LZ avec ce dictionnaire de départ que l'on notera LZ' et comparons avec la figure 11 page 20.

Cette nouvelle comparaison nous donne :
**LZ'** : /W, ED, /WE, /WEE, /WET, /WEB
**LZW** : /W, ED, /WE, /WEE, /WET, WEB,WE,D/,E/,E/W,B/
C'est-à-dire presque le double.
Notons que pour /WEB, on a, dans LZW , /W avec W qui peut être continué par E et B donc ce mot n'est pas dans le dictionnaire mais peut apparaître dans la reconstruction.

| LZ' | | | LZW | | |
|---|---|---|---|---|---|
| **Entrée** | **Connu** | **Dictionnaire** | **Entrée** | **Connu** | **Dictionnaire** |
| / | Oui | | / | Oui | |
| /W | Non | /W | /W | Non | /W |
| E | Oui | | WE | Non | WE |
| ED | Non | ED | ED | Non | ED |
| / | Oui | | D/ | Non | D/ |
| /W | Oui | | /W | Oui | |
| /WE | Non | /WE | /WE | Non | /WE |
| / | Oui | | E/ | Non | E/ |
| /W | Oui | | /W | Oui | |
| /WE | Oui | | /WE | Oui | |
| /WEE | Non | /WEE | /WEE | Non | /WEE |
| / | Oui | | E/ | Oui | |
| /W | Oui | | E/W | Non | E/W |
| /WE | Oui | | WE | Oui | |
| /WEB | Non | /WEB | WEB | Non | WEB |
| / | Oui | | B/ | Non | B/ |
| /W | Oui | | /W | Oui | |
| /WE | Oui | | /WE | Oui | |
| /WET | Non | /WET | /WET | Non | /WET |

FIG. 11 – Comparaison LZ' vs LZW

**Un dictionnaire au départ...**
On peut se poser la question de l'utilité d'un tel dictionnaire a priori en musique. En effet, cela donnerait la possibilité à l'ordinateur d'utiliser des séquences qui ne sont peut-être pas apparues dans l'original. Cette question est intéressante : l'ordinateur doit-il utiliser les éléments déjà entendus dans l'original ou peut-il conserver des dessins mélodiques (suite d'intervalles) qu'il appliquerait à la tonalité choisie ? On se heurte ici à un gros problème, car on risque l'explosion des motifs toujours croissants ou décroissants qui amènerait à un débordement de la pile. Nous utiliserons d'autres méthodes pour développer ce point.

– *Compression LZW*
Revenons un peu plus en détail sur la définition de la compression LZW, trouvée sur la page [4].

Comme déjà dit précédemment, la compression LZW est une exten-

sion de l'algorithme de Huffman. Elle consiste à trouver de manière dynamique, les motifs répétés de longueur quelconque et à coder efficacement les plus fréquents. On parle alors de codage de type dictionnaire.

Ici, la compression se fait en un passage et peut donc se faire à la volée, en "direct". On utilise aussi une structure d'arbre qui n'a pas à être enregistrée (elle est cachée de manière implicite dans le fichier compressé ! ).

La phase de lecture et le codage sont ici simultanés : lors de la lecture, on construit l'arbre au fur et à mesure des caractères rencontrés. Si la branche que l'on souhaite atteindre a déjà été créée, on s'y déplace et on lit réellement le caractère (en passant au suivant). Dans ce cas là, rien n'est enregistré dans le fichier de sortie. A l'inverse, si la branche à atteindre n'est pas présente, on la crée et on rejoint la racine de l'arbre (en consultant le caractère mais sans incrémenter la position courante). Alors seulement, la branche sur laquelle on se trouvait est codée et inscrite dans le fichier de sortie (cette branche décrit tout le parcours précédemment effectué). De plus, on crée alors la branche qui nous manquait (pour le codage éventuel de la même chaîne).

La lecture du fichier compressé suffit à reconstituer l'arbre et à décoder le fichier. En pratique, on limite la taille de l'arbre à une valeur prédéterminée et on recommence un nouveau codage quand la taille de l'arbre est dépassée.

Avantages
Ce type de codage est très efficace, particulièrement pour la compression des fichiers exécutables et des images comportant des aplats de couleur (logos, dessins).

Inconvénients
On ne peux pas prévoir la quantité d'information que va prendre en compte LZW . Cela peut ralentir le processus et ne pas être très adapté à de données lourdes. Il est bien adapté pour les textes mais peu pour l'image. Le midi serait une donnée intéressante à traiter pour envisager une analyse séquentielle, toujours dans l'optique d'un dialogue avec un instrumentiste. Mais, le fait que LZ n'utilise pas du tout de probabilité et qu'il remette tous les patterns au même niveau n'est pas une bonne chose. LZ ne permet pas de voir les redondances d'une manière globale : il repère les successions de notes mais pas les

redondances de motifs. De plus, il ne voit que l'exacte répétition. Si deux phrases sont analysées avec un détail qui les sépare, l'algorithme les considérera comme deux patterns distincts. Il créera donc une nouvelle branche dans l'arbre, occupant encore de l'espace mémoire, cela n'apportant rien d'innovant pour le parcours de l'arbre lors de la génération.

### III.2.2 PST

Pour généraliser aux arbres de suffixes, nous envisageons donc une autre méthode, celle des PST : Probabilistic Suffix Tree.
Le principe est de restreindre le dictionnaire aux éléments apparaissant un nombre significatif de fois dans tout l'extrait, ce qui implique donc des pertes avec cette méthode.
PST est un arbre non complet où le degré des nœuds varient de 0 à la taille de l'alphabet. Chaque nœud est étiqueté par un élément de l'alphabet. Tout l'alphabet est finalement représenté et chaque élément n'est présent qu'une seule fois. L'étiquette d'un nœud est donnée par le chemin parcouru pour accéder à ce nœud.

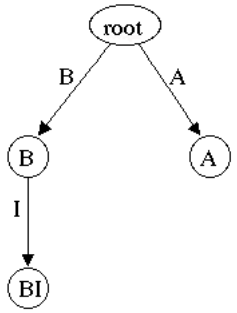


FIG. 12 – Représentation en Probabilistic Suffix Tree

Il faut ajouter à cela les probabilités. A chaque nœud est associé un vecteur de probabilité. Par exemple, dans le cas précédent, l'alphabet est A, B, I. Les vecteurs de probabilité auront donc la forme (a,b,i). Cela signifie que, en prenant X notre nœud, la probabilité de trouver A après une chaine dont le plus grand suffixe dans l'arbre est X, sera a. De même pour B, la probabilité sera b.

L'arbre devient :

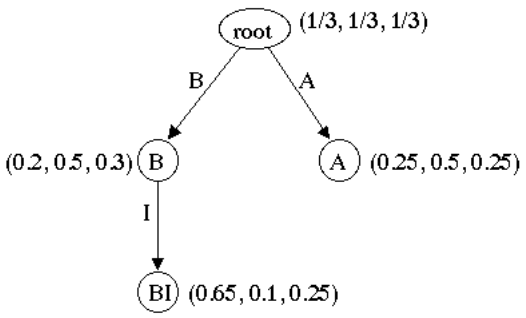


FIG. 13 – PST avec les probabilités

Par exemple, il est très peu probable (0.1) d'avoir B après BI.

Pour construire un PST, il faut donc d'abord connaître l'alphabet. Ensuite, il faut définir L, la longueur de la chaine maximale dans l'arbre. On va donc s'intéresser à toutes les sous-chaines de longueur au maximum L. On fixe également un seuil Pmin de probabilité : si une probabilité est inférieure à Pmin, la chaîne sera considérée comme négligeable. On peut choisir la valeur de Pmin selon les utilisations.
Au départ, l'arbre n'est composé que d'un nœud appelé racine. A chaque pas, donc pour chaque sous-chaine, on recherche dans l'arbre si les éléments qui sont susceptibles de "suivre" cette chaine ont une probabilité supérieure à Pmin, c'est-à-dire s'il existe une continuation non négligeable de la chaine. On fait de même avec la chaine 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 le plan statistique. Ceci constitura une des limites de cet algorithme quant à son application à la musique. Cet algorithme est donc avec pertes.

Revenons à l'attribution de probabilités. La probabilité que X suive Y est le rapport du nombre de fois que X suit Y , sur le nombre total d'occurences de Y.

Ex : aabaaabab P(b/a) = 3/6. La probabilité de trouver b après a (b sachant a) est de 3 (on trouve 3 fois ab) sur 6 (on a 6 a qui aurait pu être suivis de b).

P(b/aa) = 2/3. (2 fois aab pour 3 fois aa)

Si on veut la probabilité de b sachant a ou aa, on multiplie P(b/a) et P(b/aa).

On étudie les chaines caractère par caractère donc on aura pour "musique" :

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

Conclusion

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 forcément 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. Les figures de style qui enrichissent le morceau seront délaissées. C'est un problème pour notre travail.

PST est plus longue et plus lourde que LZ car on passe déjà une étape à calculer les probabilités et à les attribuer avant de comparer chaque unité une à une dans l'arbre. Cela fait un coût de calcul élevé et une réalisation en temps réel qui semble difficile.

### III.2.3 Ukkonen

Après avoir constaté les limites de LZ et PST pour une application à la musique, nous avons étudié l'algorithme de Mc Creight[27], qui nous a mené à celui d'Ukkonen.

1. Un algorithme séquentiel

   Le gros avantage de l'algorithme d'Ukkonen est d'être séquentiel : il n'est pas nécessaire de connaître tout l'objet pour commencer à le traiter. C'est une bonne chose en vue d'une exécution temps réel avec un instrumentiste.

   On note ST(x) (Suffix Tree) l'arbre des suffixes du mot x. Pour un mot donné, l'algorithme d'Ukkonen revient à insérer tour à tour les préfixes du mot dans l'arbre. En effet, on n'a pas besoin d'attendre la fin du mot pour commencer l'analyse. Pour le mot "abracadabra", on crée les arbres intermédiaires ST(a), ST(ab), ST(abr)..., ST(abracadabra). Cela revient à une mise à jour de l'arbre à chaque nouveau pas : on insère une lettre au(x) bon(s) endroit(s).

   Lors de la construction, Ukkonen rajoute une racine « générale », nommée

$\perp$ , reliée à la racine R par un arc ($\perp$, a, R) pour tout a de l'alphabet S.
Deux petites définitions :
Lien suffixe : pour chaque noeud interne de l'arbre représentant locus de ax, on peut rajouter un lien vers le locus de x.
Locus de x : noeud à la fin du chemin représentant ce mot.

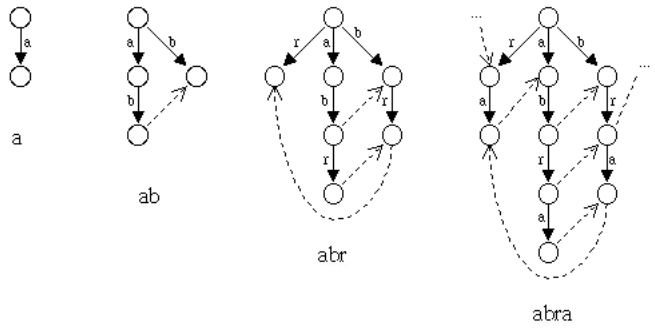


FIG. 14 – Premières étapes d'Ukkonen sur abra

On commence par s'occuper du ST (Suffix Tree). A l'étape i, on a déjà créé le STi-1.
Définition : chemin suffixe : suite de nœuds p1,p2,..,pi avec p1 le locus de x1... xi-1 jusqu'à pi = R.
Exemple : "abracadabra" à l'étape 6 : on a déjà le ST(abrac) et on veut placer a. On a p1 = abrac , p2 = brac , p3 = rac , p4 = ac , p5 = c et p6 = R. (p1, p2, p3, p4, p5, p6) forme le chemin suffixe à l'étape 6. D'après la définition des liens suffixes , on a s(p1) = p2 , s(p2) = p3... Donc, à l'étape i , on rajoute à tout nœud pj (j de 1 à i) un arc (pj, xi, q) , q nouveau nœud s'il n'existait pas. Or, si on en trouve un déjà existant, tous les pj suivants seront idem donc on peut passer à l'étape suivante.

2. Description de l'algorithme
Top est p1, p est le noeud de parcours du chemin suffixe, oldp le noeud précédement créé dans la remontée de ce chemin.

```
Initialiser : R ,⊥, s( R) = ⊥
Pour tout a : créer arc (⊥, a, R)
Top reçoit R
Tant que (xi <> dollar) cad tt que pas fin du mot, faire
        P reçoit top
        Tt que le nœud qui suit p par xi n'existe pas
                Créer q et arc (p, xi , q)
                Si (p <> top) alors s(oldp) reçoit q
                Oldp reçoit q
                P reçoit s(p)
        Fin tant que
        S(oldp) reçoit nœud (p,xi-]
        Top reçoit nœud (top, xi]
Fin tant que
Fin
```

FIG. 15 – Algorithme d'Ukkonen

| **Etape** | **Mot** | **Xi** | **P** | | **S(p)** | **Q** | **Top** | **Oldp** | **S(oldp)** |
|---|---|---|---|---|---|---|---|---|---|
| 0 | abraca | | | | | | R | | |
| 1 | abraca | a | R | R | R | ( 1 ) | ( 1 ) | ( 1 ) | ( 1 ) = R |
| 2 | braca | b | ( 1 ) | R | R | ( 2 ) | | ( 2 ) | ( 3 ) |
| 2 | braca | b | R | R | R | ( 3 ) | ( 2 ) | ( 3 ) | ( 3 ) = R |
| 3 | raca | r | ( 2 ) | ( 3 ) | ( 3 ) | ( 4 ) | | ( 4 ) | ( 5 ) |
| 3 | raca | r | ( 3 ) | R | R | ( 5 ) | | ( 5 ) | ( 6 ) |
| 3 | raca | r | R | R | R | ( 6 ) | ( 4 ) | ( 6 ) | ( 6 ) = R |
| 4 | aca | a | ( 4 ) | ( 5 ) | ( 5 ) | ( 7 ) | | ( 7 ) | ( 8 ) |
| 4 | aca | a | ( 5 ) | ( 6 ) | ( 6 ) | ( 8 ) | | ( 8 ) | ( 8 ) = R |
| 4 | aca | a | ( 6 ) | R | R | ( 9 ) | ( 7 ) | ( 9 ) | ( 1 ) |
| 5 | ca | c | ( 7 ) | ( 8 ) | ( 8 ) | ( 10 ) | | ( 10 ) | ( 11 ) |
| 5 | ca | c | ( 8 ) | ( 9 ) | ( 9 ) | ( 11 ) | | ( 11) | ( 12 ) |
| 5 | ca | c | ( 9 ) | ( 1 ) | ( 1 ) | ( 12 ) | | ( 12 ) | ( 13 ) |
| 5 | ca | c | ( 1 ) | ( 1 ) | ( 1 ) | ( 13 ) | ( 10 ) | ( 13 ) | ( 13 ) = R |
| 6 | a | a | (10) | (11) | (11) | ( 14 ) | | ( 14 ) | ( 15 ) |
| 6 | a | a | (11) | (12) | (12) | ( 15 ) | | ( 15 ) | ( 16 ) |
| 6 | a | a | (12) | (13) | (13) | ( 16 ) | | ( 16 ) | ( 17 ) |
| 6 | a | a | (13) | (13) | (13) | ( 17 ) | ( 14 ) | ( 17 ) | ( 17 ) |

FIG. 16 – Déroulement de l'algorithme d'Ukkonen sur abraca

Si on fait tourner l'algorithme sur le mot abracadabra, on a le tableau

précédent.

On peut constater que 17 noeuds sont nécessaires, ce qui est presque le triple du nombre de lettres. De plus, pour construire l'arbre « on-line » avec Ukkonen, la programmation est fastidieuse et les calculs trop lourds. Certes, le traitement commence dès que des données sont présentées, donc, sans attendre la fin du fichier mais, le modèle n'est pas assez puissant pour un projet temps réel.

# IV L'oracle des facteurs

On fait facilement le lien entre la modélisation d'une pièce de musique et celle d'un segment d'ADN. En effet, ce sont deux domaines où l'on utilise les arbres de suffixes, pour trouver les répétitions, détecter les patterns importants. Dans les deux cas, on sait que les données sont lourdes et qu'il faut les représenter dans un espace mémoire le plus faible possible. Pour cela, l'automate semble être un modèle très efficace au point de vue computationnel. De plus, Allauzen, Crochemore et Raffinot [3] donnent une version totalement incrémentale et efficace d'un automate particulier appelé « l'oracle des facteurs », utilisé pour les recherches sur le génome.
Après avoir rappeler ce qu'est un automate, nous expliquerons en détail le principe de l'oracle des facteurs et la façon dont nous l'avons programmé pour Open Music.

## IV.1 Les automates

### IV.1.1 Définition

Un automate fini est défini comme un quintuplet A, Q, I, F, T où :

A 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.

On part d'un état initial, on suit les règles de transition. Plusieurs situations peuvent se présenter :

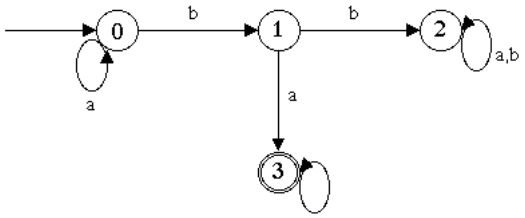


FIG. 17 – Automate n°1

Ici, l'état initial est (0). L'état final est (3).L'alphabet est a ,b.
Les règles sont :

Je suis en 0 : je reste en 0 par a et je passe en 1 par b .
Je suis en 1 : je passe en 2 par b et je passe en 3 par a.
Je suis en 2 : je reste en 2 quelque soit la nouvelle entrée.
Je suis en 3 : je reste en 3 quelque soit la nouvelle entrée.

Donc, l'ensemble des règles est :
(0,a,0), (0,b,1), (1,b,2), (1,a,3), (2,(a/b),2), (3,(a/b),3).

Prenons l'exemple de plusieurs mots :
aba : liste des états : 0-0-1-3. Je finis en 3 donc le mot est accepté.
baba : 0-1-3-3-3 : mot accepté
abba : 0-0-1-2-2 : mot refusé : je termine en 2 qui n'est pas un état final.
On dit que le mot n'est pas reconnu par l'automate.
On déduit la grammaire reconnue par cet automate : a*ba(a/b)*
*On rappelle que * signifie 0 ou plus. On utilisera + pour signifier 1 ou plus.*

On dit qu'un automate est déterministe quand il y a pour chaque état, au plus une flèche qui part pour chaque lettre. Une grammaire est dite régulière s'il existe un automate fini dont l'ensemble des mots reconnus est exactement celui engendré par la grammaire.

### IV.1.2 Comparaison Arbre / Automate

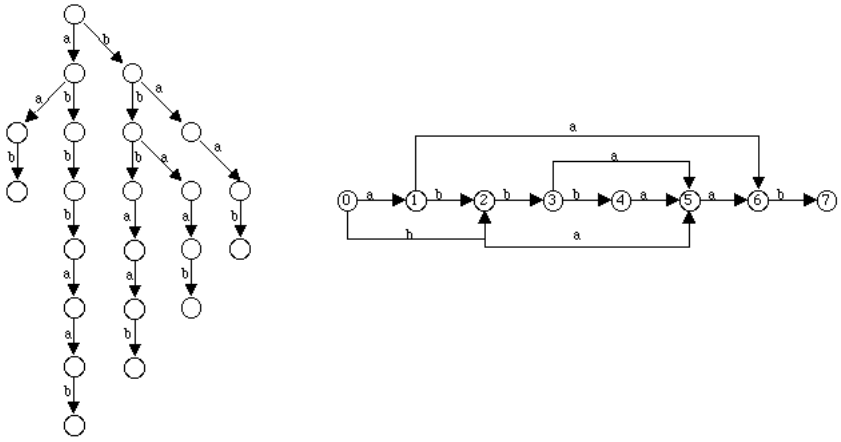

FIG. 18 – Arbre (à gauche) vs Automate (à droite) du mot abbbaab

On peut voir ici que l'arbre nécessite 17 états et 21 flèches alors que pour la représentation du même mot, l'automate nécessite 8 états et 11 flèches. Cela prouve bien que la taille de l'espace mémoire requise est optimisée avec l'automate. L'automate est donc un arbre "aplati".

## IV.2 Oracle des facteurs

### IV.2.1 Définition

L'oracle des facteurs est un automate qui reconnaît tous les facteurs d'un mot p1... pm. Cet automate est acyclique et surtout linéaire en temps et en espace. L'avantage de l'oracle des facteurs est que le nombre d'états créés est minimal, donc, optimal quant à l'espace mémoire requis. Un autre point important de cet algorithme est qu'il est simple d'implémentation.

1. Qu'est-ce q'un facteur
   Reprenons la définition donnée dans l'article (Allauzen, Crochemore, Raffinaut)[3]. Un mot x $\in \sum^*$ est un facteur de p si p peut être écrit p = uxv, u, v x $\in \sum^*$.
2. Le principe
   Pour un mot de m lettres, on construit m+1 états, numérotés de 0 à m.
   On relie chaque état de 0 à m à son suivant par la lettre correspondante pour former l'automate du mot.
   Par exemple, l'automate de " abbbaab " est :

   (0)–a–>(1)–b–>(2)–b–>(3)–b–>(4)–a–>(5)–a–>(6)–b–>(7)

   Ensuite, pour chaque état de 0 à m, on cherche si on peut accéder , par une lettre de l'alphabet différente de celle de l'arc sortant existant, au suffixe d'un facteur f de p , c'est-à-dire à un facteur de p sans sa première lettre.
   Dans ce cas, on crée une nouvelle flèche qui part de l'état courant vers le suffixe de f, par le préfixe f.
   Ainsi, on peut accéder à tous les facteurs du mot.
   Voici un exemple.
   Nous montrerons ensuite comment implémenter cet algorithme d'une façon séquentielle, ce qui est particulièrement intéressant .
   Continuons avec le même exemple : "abbbaab".
   L'oracle des facteurs est alors :

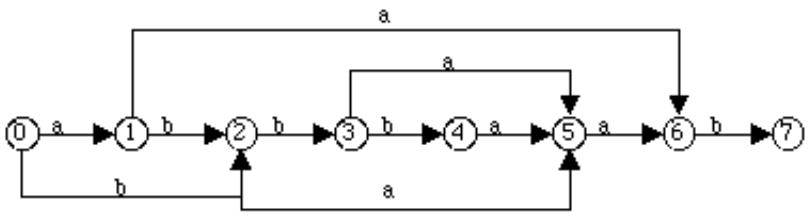


FIG. 19 – Oracle de abbbaab

Si on prend la liste des facteurs de " abbbaab ", on a :
abbbaab, bbbaab, bbaab, baab, aab, ab, b pour les suffixes
abbbaa, abbba, abbb, abb, a, bbbaa, bbaa, baa, aa, bbba, bba, ba, bbb, bb pour les autres facteurs.
On peut tous les retrouver dans l'oracle créé ci-dessus.

| **Facteur** | **Chemin** | **Facteur** | **Chemin** |
|---|---|---|---|
| abbbaab | 0-1-2-3-4-5-6-7 | baab | 0-2-5-6-7 |
| abbbaa | 0-1-2-3-4-5-6 | bbbaa | 0-2-3-4-5-6 |
| abbba | 0-1-2-3-4-5 | bbaa | 0-2-3-5-6 |
| abbb | 0-1-2-3-4 | baa | 0-2-5-6 |
| abb | 0-1-2-3 | bbba | 0-2-3-4-5 |
| aab | 0-1-6-7 | bba | 0-2-3-5 |
| ab | 0-1-2 | ba | 0-2-5 |
| aa | 0-1-6 | bbb | 0-2-3-4 |
| a | 0-1 | bb | 0-2-3 |
| bbbaab | 0-2-3-4-5-6-7 | b | 0-2 |
| bbaab | 0-2-3-5-6-7 | | |

FIG. 20 – Facteur et chemin dans l'oracle

Remarque
A l'inverse de PST qui supprime les éléments considérés comme peu important, l'oracle garde tous les facteurs de l'objet traité. Et même plus : il crée de nouveaux patterns. Cela pourrait être gênant dans certaines expériences, mais pour notre étude, il est intéressant de multiplier les continuations. Par exemple, dans le schéma précédent, si on suit le chemin 0-1-2-5-6-7, alors, on arrive à l'état final en lisant le mot abaab, qui n'est pas un facteur de abbbaab. Le fait de multiplier les solutions pour le parcours de l'arbre est pour nous intéressant et évitera à la machine de tomber dans des états puits ou isolés.

Ce qui nous intéresse particulièrement est l'algorithme séquentiel de construction de l'automate. Cela signifie que l'on construit pas à pas, en lisant les lettres une à une de droite à gauche. Cela veut dire que l'on n'est pas obligé de connaître tout le mot qui va être traité avant de commencer son analyse. C'est très intéressant dans une optique de " dialogue " entre un musicien et une machine : le musicien commence à jouer et la machine analyse au fur et à mesure, sans savoir ce qui va se passer et génère une improvisation quand c'est son tour. Nous allons donc essayer de décrire l'algorithme séquentiel de construction de l'Oracle des facteurs.

### IV.2.2 Algorithme

Au départ, on initialise l'automate en créant un état numéroté 0, de suppléance égale à –1, que l'on note Sp(0) = -1. Nous reviendrons sur cette notions de suppléance par la suite. Puis, à chaque lettre du mot, on l'ajoute comme ceci : On donne en entrée le mot déjà traité (n lettres) et la nouvelle lettre $\sigma$. On crée un nouvel état à qui l'on donne le numéro (n+1). On relie l'état n à l'état (n+1) par $\sigma$. La variable k prend la valeur de Sp(n). Pour la première lettre par exemple, n = 0 donc k prend –1. Et tant que k est supérieur à –1, et qu'il n'y a pas de flèche sortante de k par $\sigma$, alors, on crée cette flèche vers l'état (n + 1) et k prend la suppléance de k. Au premier tour, k = -1 donc on ne rentre pas dans cette boucle. Cela est normal puisqu'il n'y a alors que deux états donc, une seule flèche possible. Pour finir, on attribue une valeur à Sp(n+1) qui est soit 0 si k = -1 au sortir de la boucle et l'arrivée de la transition de l'état k par $\sigma$ sinon. La suppléance va permettre de retrouver tous les facteurs du mot.

On renouvelle cette étape à chaque nouvelle lettre qui se présente.

L'oracle est utile pour calculer des répétitions afin de comparer de très longs mots tels que des séquences biologiques. Le calcul des répétitions d'un mot mène naturellement à établir une factorisation du mot qui donne un schéma séquentiel de compression. Il est également possible d'obtenir une méthode globale de compression en utilisant des règles de grammaire qui décrivent le mot.

## IV.3 Programmation

### IV.3.1 La classe Oracle

On définit une classe oracle, qui représentera l'automate créé au fur et à mesure du déroulement de l'algorithme. Cette classe contient plusieurs champs :

- Vectext
  On représente l'automate par le mot traité pour le former, champs que l'on note « vectext ». Par exemple, prenons l'automate suivant :

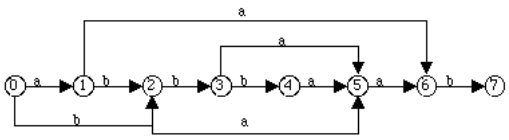


FIG. 21 – Vectext = abbbaab

  Son Vectext sera abbbaab.

- hashtransition
  C'est un tableau, regroupant toutes les transitions partant du noeud dont on parle. Ces transitions sont représentées par des couples (destination, lettre de transition). Par exemple, si on reprend l'exemple précédent, dans les hashtransitions du noeud 3, on trouvera (4,b) et (5,a).

- hashsuppl
  Pour chaque état, on note dans ce champs sa suppléance.
  Reprenons cette définition à la base, avec un exemple : babaab. Supposons ce mot comme un préfixe de p, de longueur 7. On cherche dans ce mot le plus long suffixe qui est répété au moins deux fois. On trouve ab, qui a deux occurences dans ce mot : l'une en 2, l'autre en 7. Alors, on dit que 7 a pour suppléance 2. En effet, si l'on imagine que 7 peut être suivi de b par exemple, alors, on sait que la continuation de ab peut être b (par 7) et a (par 2). Ainsi on élargit les possibilités de parcours de l'automate.

- maxetat
  Comme son nom l'indique, ce champs contient le numéro de l'état maxi-

mum de l'oracle.

- nbresauts
  Cet attribut va servir à calculer la fidélité de ce que produit l'algorithme avec le morceau de départ. Lors du parcours de l'automate, on va incrémenter cette données si un état n'est pas suivi de l'état portant le numéro suivant. Par exemple, on se situe dans l'état 15 et le choix est fait pour l'état 23. Alors, on ne suit pas ce qu'avait fait l'original donc, on s'en écarte. On comptabilise cela comme un saut.

- ptitsaut
  C'est ici le même principe mais, on incrémente cette donnée uniquement si le noeud d'arrivée du saut est proche du fils du noeud de départ. Si le noeud 23 ressemble au noeud 16 qui était le fils direct, alors, on reste fidèle à l'original donc, on compte cela comme un petit saut. On se servira ensuite de ces données dans les fonctions Fid et Fidelite.

### IV.3.2 Les fonctions de la classe Oracle

Il y a de nombreuses fonctions qui se rapportent à la classe Oracle. Nous allons les énumérer et en donner une brève description.

- initialize-instance
  Permet de créer une nouvelle instance de la classe Oracle et d'en initialiser les champs.

- transition
  Cette méthode prend en paramètres un oracle, un état et un objet. L'opération consiste à rechercher dans la table des transitions de l'oracle le couple (état,objet) et de retourner la solution : elle permet donc de savoir ou va la flèche qui part de « état » par la transition « objet » dans l'automate « oracle ».

- suppleance
  Méthode d'accès au champs « suppleance » de l'oracle.

- text
  Retourne le mot qui a formé l'oracle. C'est la méthode d'accès au champs « vectext » de l'oracle.

### IV.3.3 La classe Pythie

A partir de la classe Oracle, on crée une autre classe à laquelle on rajoute un champs à ceux hérités : comparateur. En effet, au cours de la construction de l'Oracle, on va comparer les accords deux à deux pour les « classer ». S'ils sont considérés comme égaux, alors, on les classe dans le même état Dans le cas contraire, on crée un nouvel état.
Nous avons défini plusieurs fonction de comparaison que nous décrirons par la suite. On attribue à ce nouveau champs « comparaison » la fonction choisie pour l'opération.

### IV.3.4 Les fonctions de la classe Pythie

Il faut redéfinir certaines fonctions ou les surdéfinir. On doit également en créer de nouvelles, comme l'affichage de l'automate.

- initialize-instance
  Même chose que pour la classe Oracle.

- transition
  Permet de vérifier qu'il y ait un transition partant de « état » par « objet » dans « pythie ».
  On va rechercher dans l'oracle la table des transitions. Puis, on ne garde que celles qui partent de l'état donné en argument. Ensuite, on compare par la méthode choisie (dans le champs comparateur) toutes les transitions sortantes à l'objet donné en argument. On la renvoie ou bien, on retourne nil.

- flink
  Méthode d'accès à la table de transitions d'une instance de la classe pythie.

- fsuppl
  Méthode d'accès à la table de suppléances d'une instance de la classe pythie.

### IV.3.5 Les méthodes de comparaison

Nous avons programmé 7 méthodes de comparaisons entre deux accords. Nous allons les décrire une à une, après avoir décrit la façon dont sont

représentés les accords.

1. Représentation des accords
   Grâce à la méthode « Midi->Cross », programmée par Olivier Lartillot, on passe de la représentation midi du morceau à une représentation en liste d'accord exploitable par notre algorithme. Nous ne nous attarderons pas sur cette fonction mais plus sur la représentation après cette opération qui retourne donc une liste d'accords. Chaque accord est représenté par la liste des notes, associée à sa durée, chaque note étant représentée par un couple (hauteur vélocité).
   Par exemple, un accord de 4 sons, on aura :

   (((h1 v1)(h2 v2)(h3 v3)(h4 v4))) durée)

   Les sons sont classés par ordre croissant donc, (h1 v1) représente la note la plus grave et (h4 v4) la plus aigue. On peut travailler sur plusieurs canaux donc, il faut rajouter un niveau de parenthèses.
   Ainsi, pour chaque fonction de comparaison, on commencera par récupérer l'accord, donc la liste des hauteurs classées.
   On prend le premier élément de la liste donc pour notre exemple « (((h1 v1)(h2 v2)(h3 v3)(h4 v4))) ». Pour chaque canal, on récupère les accords. Dans notre exemple, nous n'avons qu'un canal avec un seul accord. On récupère donc ((h1 v1)(h2 v2)(h3 v3)(h4 v4)). Pour chaque note de cet accord, on prend sa hauteur. On a donc au final la liste des hauteurs de l'accord.

2. Méthode de comparaison n°1 : basse-intersection-durée
   Cette première méthode s'établit sur la comparaison des notes les plus graves, donc les premières des « chord ». Si elles ne sont pas égales, on compare l'intersection des deux accords ramenés à une seule octave. En dernier lieu, on compare les durées des deux accords. Si celles-ci ne sont pas égales, alors, les accords sont considérés comme différents.

3. Méthode de comparaison n°2 : basse
   Dans cette méthode, on récupère la basse de chaque « chord » grâce à la fonction getbass. On les compare et les accords sont considérés comme similaires si les basses sont égales modulo 12.

4. Méthode de comparaison n°3 : Estrada
   On compare par cette méthode les textures. On regarde la distance estrada entre les deux accords.
   Reprenons la définition de cette distance Estrada, d'après un travail de

Georges Bloch sur l'algorithme mis au point par le musicologue Jolio Estrada.
On réduit chaque « chord » à une octave. On calcule les intervalles entre les différentes notes et on complémente à l'octave. Prenons l'exemple du fameux Do Majeur :
Entre Do et Mi, on a une tierce majeure, notée 4 en nombre de demi-tons.
Entre Mi et Sol, tierce mineure notée 3.
Entre Sol et Do (simulation de l'octave de la note la plus grave), on a une quarte juste notée 5.
On classe ensuite les chiffrages par ordre croissant. Do Mi Sol est donc représenté par (3 4 5).
C'est ce que fait la fonction estr-analyse, opération réalisée sur chaque accord de la comparaison.

La deuxième étape consiste à comparer le nombre d'intervalles en commun. La fonction estr-compare réalise cette comparaison en incrémentant un paramètre à chaque intervalle commun.

Dans la dernière étape, on calcule la distance qui est donnée par la formule suivante : Si les deux accords ont le même nombre de notes :

Distance = nombre de notes - nombres d'intervalles communs - 1
Si Distance = -1, alors Distance = 0

Si les deux accords ont un nombre de notes différent :

Distance = nombre max de notes - nombres d'intervalles communs - 1

On choisit ensuite dans notre fonction de comparaison où l'on met la limite. Pour nous, les accords sont considérés égaux si la distance estrada est inférieure ou égale à 1.

5. Méthode de comparaison n°4 : intersection
Cette fonction est assez simple et assez logique. Plus les accords ont de notes en commun, plus ils sont considérés comme similaires. Ainsi, après les avoir réduits à la même octave, on fait l'intersection des deux ensembles. On choisit un seuil de ressemblance et si le nombre de notes en commun est inférieure à ce seuil, on renvoie nil sinon t.

6. Méthode de comparaison n°5 : table d'accords
Le principe est de rechercher une fondamentale aux accords ; s'ils ont

la même, on les dira égaux. Pour trouver cette fondamentale, on commence par créer une base de données des accords les plus fréquents et de leur chiffrage. On note également les accords avec des notes absentes. On associe leur alors le chiffre de la fondamentale. Notons que les renversements sont neutralisés quand on remet tout sur une même octave.
Prenons l'exemple de la septième majeure. On réduit les accords sur une octave. On calcule la distance en demi-tons à la basse.Toutes les septièmes majeures sont représentées ci-dessous.
Do-Mi-Sol-Si (0 4 7 11) avec la fondamentale 0.
...
Réb-Fa-Lab-Do Do-Réb-Fa-Lab (0 1 5 8) avec la fondamentale 1.
...
Fa-La-Do-Mi Do-Mi-Fa-La (0 4 5 9) avec la fondamentale 5.
...
Lab-Do-Mib-Sol Do-Mib-Sol-Lab (0 3 7 8) avec la fondamentale 8.
On fait cette opération pour la septième majeure sans tièrce ou sans quinte.

Les accords traités dans la base sont :

(a) Accord Parfait Majeur
(b) Accord Parfait mineur
(c) Septième Majeure
(d) Septième mineure
(e) Septième de dominante
(f) Septième diminuée
(g) Quinte augmentée
(h) Neuvième + 7e Majeure
(i) Neuvième + 7e mineure
(j) Neuvième + 7e de dominante
(k) Neuvième + 7e diminuée

Pour chacun des deux accords de la comparaison, on récupère donc sa fondamentale en sélectionnant dans la base la note qui correspond à la ligne du chiffrage de l'accord. On compare ensuite ces deux fondamentales.
Pour le cas des septièmes diminuée et quintes augmentées, on traite le cas à part car ce sont des ensemble formés de mêmes intervalles ( (0

3 6 9) pour la 7e et (0 4 8) pour la 5te). On ne peut donc pas définir leur fondamentale. On prendra donc une note de l'accord comme fondamentale. Cela n'est pas très gênant car ces accords-là ont très souvent le rôle de pivot dans la structure musicale donc peuvent appartenir à plusieurs tonalités.

7. Méthode de comparaison n°6 : Hindemith
   Cette fois, nous partons d'une méthode de calcul de la basse proposée par le compositeur et musicologue Hindemith.
   Le principe est de trouver s'il y a une quinte dans l'accord. Alors, la fondamentale est la basse de la quinte. Sinon, on recommence l'opération avec la tierce majeure. En dernier lieu, la fondamentale sera la note la plus basse de l'accord.
   Pour trouver cette fondamentale, on procède donc de la sorte. Prenons la première note de l'accord trié, donc, la plus grave. On regarde les intervalles formés avec les autres notes de l'accord. Si un intervalle de quinte est trouvé, soit un chiffrage 7, la première note est nommée fondamentale de l'accord. Sinon, on continue avec la deuxième note, ainsi de suite jusqu'à la fin de chord.
   Si aucune fondamentale n'est trouvée, on recommence l'opération avec la tierce majeure, de chiffrage 4.
   Après cette opération, si la fondamentale n'a toujours pas été définie, on prendra la note la plus grave de l'accord.

8. Méthode de comparaison n°7 : Estrada + Hindemith
   Premièrement, on regarde si les accords sont considérés égaux avec la distance estrada. Si non, on recompare en utilisant cette fois la recherche de fondamentale d'Hindemith.

9. Affichage de l'oracle
   Nous avons bien sûr des fonctions d'affichage pour l'oracle, ce qui nous permet d'étudier le parcours qui a été fait, comme par exemple, vérifier si l'ordinateur a choisi ce chemin au hasard ou bien s'il n'en avait pas le choix.
   On affiche donc, grâce à la fonction du même nom, la liste des noeuds, associés chacun à la liste de leurs suivants.

### IV.3.6 La méthode om-analyse

C'est cette méthode qui regroupe et utilise toutes les fonctions décrites auparavant. Elle prend en argument le texte à analyser (ou le fichier midi) et

la fonction de comparaison choisie. Ce choix se fait graphiquement, comme le montre la figure suivante.

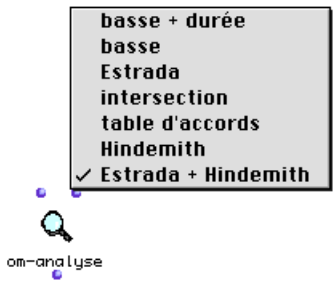


FIG. 22 – Menu déroulant pour le choix de comparaison.

C'est dans cette fonction que nous choisissons l'icone qui va représenter cette méthode. Dans le patch, elle-seule apparaitra et représentera toute l'analyse des données midi. Elle crée donc une instance de Pythie et forme, avec le comparateur sélectionné, l'oracle des facteurs du fichier en entrée, selon l'algorithme donné plus haut.

### IV.3.7 Les méthodes de parcours de l'oracle

Suite à cette analyse, à ce découpage, à cette organisation du morceau en petites parties, nous avons voulu parcourir cet automate pour « recréer un improvisation ». En fait, il ne s'agit plus d'une improvisation puisque la musique est maintenant sur un support. Nous parlerons d'une simulation. Celle-ci consiste à parcourir l'automate, en respectant le plus possible le sens musical, mais sans plagier l'original, ce qui n'aurait alors aucun intérêt. Nous avons donc programmé deux méthodes de parcours de l'arbre. La première méthode aura quelques petites variations, c'est pourquoi nous arrivons à 4 possibilités.

1. Méthode de parcours n°1
   La première méthode, comme les deux suivantes d'ailleurs, repose sur un tirage de chiffre aléatoire entre 0 et 10. Ce paramètre est appelé *chance*.
   Si *chance* est entre 0 et 3, alors on choisit le fils direct, c'est-à-dire le noeud suivant. Par exemple, si je suis dans l'état 567, je continue mon chemin en 568.
   Si *chance* est entre 4 et 7, on choisit une des flèches partant de l'état

présent dans la table des transitions. Notons que l'on peut retomber sur le noeud suivant, qui appartient à cette table.
Si *chance* est entre 7 et 10, on se dirige vers la suppléance de l'état présent. Celle-ci ne peut pas être le suivant.
Lorsque l'on arrive sur le dernier noeud, on repart au début.

Pour les méthodes 2 et 3, le principe est le même. On change juste les paramètres ou les actions suivant la valeur de "chance", pour comparer avec cette première méthode.

2. Méthode de parcours n°4
   Ici, on décide de limiter la fidélité en insérant une limite de notes qui se suivent dans l'original et qui se retrouvent dans la simulation. Ainsi, quand on fait le choix du fils direct, on incrémente une variable. Quand on fait un saut, elle revient à zéro.
   Pour chaque choix, s'il y a des suivants, on prend un chiffre au hasard entre 0 et 5.
   Si ce chiffre est 4 ou 5, ou que la limite de fidélité est atteinte, on choisit un état éloigné de l'état présent, en retirant son suivant de la liste. Il se peut d'ailleurs que le fils direct soit le seul élément de la table de continuation, auquel cas, on prend la suppléance comme état suivant.
   Sinon, dans le cas d'un chiffre tiré entre 0,1,2,3 ou 4, on choisit le fils direct, si la limite n'est pas atteinte bien sûr.

### IV.3.8 La méthode om-improvise

Dans cette méthode, on choisit la fonction de parcours de l'oracle que l'on veut. On donne en entrée, l'oracle créé par la fonction om-analyse, le nombre de note que l'on veut créer et la fonction browse, choisie parmi les 4 dans un menu déroulant.

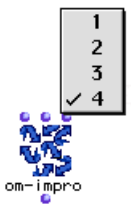


FIG. 23 – Menu déroulant pour le choix du parcours

En sortie, on relie cette boîte om-impro à un chordseq pour récupérer la simulation.

## IV.4 Bilan

### IV.4.1 Résultats obtenus

Voici un exemple de patch Open Music utilisant les méthodes décrites ci-dessus. Le fichier midi est traité dans « midi->cross », avant d'être analysé par « om-analyse ». L'oracle créé est ensuite donné en entrée à la méthode « om-improvise » qui va le parcourir et sortir une simulation récupérée dans un chordseq.

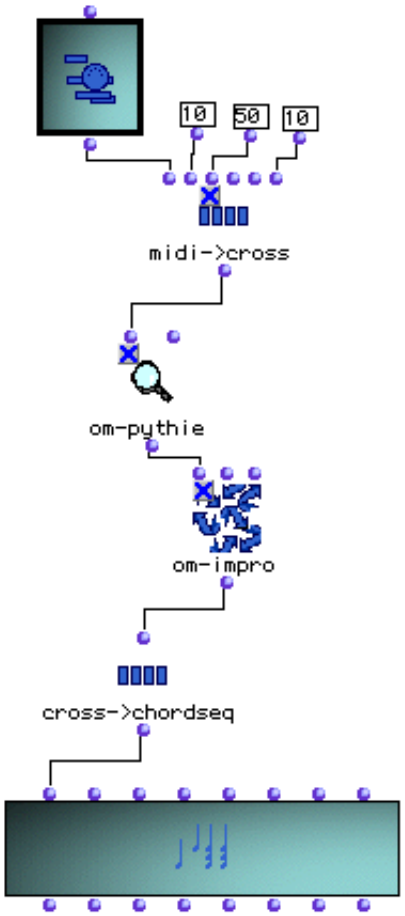


FIG. 24 – Patch Open Music utilisant la librairie Oracle

Nous avons constaté que, lorsque l'on suit l'automate par les flèches de facteurs et les flèches de suppléance, tout en générant une nouvelle séquence, nous étions dans une situation absolument équivalente à celle qui consiste à analyser dans la séquence engendrée le plus long contexte, et à prendre une décision de continuation selon ce contexte. Or, cette analyse du plus long contexte est très coûteuse et elle est en quelques sortes donnée gratuitement de part la structure de l'oracle des facteurs. C'est donc un argument supplémentaire en faveur de notre choix.
Il est à noter que cette caractéristiquetrès importante pour nous n'est pas décrite dans l'article original de Allauzen, Crochemore et Raffinaut [3].

### IV.4.2 Description de la crédibilité

Malgré parfois quelques modulations et cassures grossières, l'ensemble des résultats est apparu comme crédible. Cependant, il nous a fallu nous assurer que ce qui nous semblait être un bon résultat musical n'était pas tout simplement une copie de l'original. Pour cela, nous avons créée une fonction de fidélité, retournant le pourcentage de similitude entre le fichier original et la solution de l'algorithme.

A l'écoute, il ne nous semble pas impossible que puisse naître une confusion chez les auditeurs entre un musicien et une simulation. Nous avons voulu valider plus sérieusement ces simulations en procédant à une série d'expériences d'écoute.

# V L'expérimentation

## V.1 Les motivations de l'expérience

C'est sur le modèle du test de Turing que nous voulons mener notre expérience. Le test de Turing se résume à une expérience dans laquelle un homme est mis en parallèle avec une machine et l'on demande à un observateur n'ayant accès qu'aux messages échangés, de différencier l'homme de la machine. Turing était convaincu, en 1948, que tout n'était qu'un problème d'information et que le développement des technologies permettrait d'ici cinquante ans aux machines de tenir en échec l'être humain au moins cinq minutes.
C'est donc sur ce test qu'est basé le principe de notre expérience, décrit dans la partie suivante. Voyons tout d'abord les objectifs de cette expérimentation, c'est-à-dire les motivations qui nous ont poussés à les concrétiser. Trois intérêts majeurs ont retenu notre attention :

- la validation de l'algorithme
  C'est le premier objectif. On désire connaître la réaction de sujets face à l'algorithme, savoir s'il est proche d'un humain.

- l'amélioration de l'algorithme
  Ces données expérimentales peuvent potentiellement offrir un retour (feedback) sur les points faibles de l'algorithme : si certains éléments déclenchent la même réaction négative chez les sujets, nous saurons où travailler pour l'améliorer.

- intérêt psychoacoustique
  Cette expérience peut avoir un grand intérêt pour comprendre le processus de catégorisation.

On se concentrera sur le premier point dans l'étude des résultats expérimentaux.

## V.2 Préparation des expériences

### V.2.1 Choix du style

Notre étude porte sur l'improvisation. Il nous a donc paru naturel de choisir le style musical dans lequel l'improvisation a été développée le plus : le jazz. Il n'est bien sûr pas le seul, mais c'est certainement le plus proche de nous, de par ses origines et ses revendications.
Notre algorithme analyse et génère une simulation à partir d'un fichier midi. Il nous fallait donc une base de données d'improvisations jazz en midi. Pour

cela, deux pianistes ont répondu à notre invitation et ont donc improvisé pendant environ une heure chacun au studio 5 de l'IRCAM.

### V.2.2 Concerts live in IRCAM

L'ambiance concert nous a semblé être une mise en situation nécessaire pour les musiciens, plutôt que de les enregistrer seuls.
Pascal Borron, professeur de piano au Conservatoire National de Région de Dijon et Bernard Magnien, professeur de piano à l'école de jazz Jazz'on de cette même ville, ont chacun un style qui leur est propre, le premier plutôt inspiré de Chick Corea et le second de Bill Evans. Pour notre étude, cela va nous permettre de tester si notre algorithme est capable de capturer les deux styles, ou, s'il capte mieux l'un des deux, auquel cas il pourrait y avoir confusion entre un pianiste A et son clone, chose qui ne se produirait pas pour le pianiste B.
Les deux pianistes ont choisi des thèmes, 11 pour Borron et 8 pour Magnien dont 3 en commun :*All the things you are, Autumn Leaves(variation pour Borron), Caravan.* Les autres thèmes choisis par Borron sont *Whisper note, Blusette, Besame Mucho, Someday my prince will come, Corcovado, Lullaby of Birdland, (...)* et ceux par Magnien *Alone together, Misty, Stella by Starligth, My romance, Beautiful Love.* L'intérêt de ces thèmes est qu'ils sont tous issus du Real Book, sorte de bible de standards de jazz, et qu'ils sont comparables sur le plan des progressions harmoniques ou rythmiques. Ils sont donc extrêmement cohérents entre eux, ce qui permet à notre étude de pouvoir être testée sans problème sur des morceaux différents.
Ces deux musiciens ont bien sûr joué sur un piano midi, dans le but de récupérer deux heures de "matériau" pour le programme. Parallèlement, quatre micros ont été placés pour permettre de récupérer les données audio, ce qui a permis de remercier les pianistes par un disque de leur prestation.

## V.3 Description de l'expérience

### V.3.1 Les sujets

Les sujets étaient au nombre de 12, agés entre 23 et 35 ans. Ce sont tous des musiciens, amateurs plus ou moins confirmés de jazz. Certains avaient assisté aux concerts de Pascal Borron et Bernard Magnien. Les sujets ne sont donc pas néophites en la matière.

### V.3.2 Les stimuli

Pour chaque morceau, nous avons sélectionné dans Digital Performer, des extraits de l'original, et de sa simulation, c'est-à-dire après le traitement de l'oracle et de la génération. Ces extraits sont donc des données midi. Pour enlever l'aspect mécanique typique d'un fichier midi, nous avons passé chaque extrait d'une minute maximum dans le sampleur Unity avec des sons de "Grand Piano" (Yamaha).

### V.3.3 La procédure

1. Phase 1 : phase d'apprentissage
   Le but de cette phase est que les sujets fassent la différence entre les deux styles pianistiques proposés. Pour cela, ils disposent de 10 extraits de 45 secondes, (5 de Magnien, 5 de Borron) qu'ils écoutent une seule fois, dans l'ordre désiré. Ensuite, pour vérifier qu'ils font la distinction des styles, une dizaine d'extraits sont passés et ils doivent deviner qui en est l'interprète. Après chaque extrait, on leur donne une confirmation si la réponse est bonne ou la réponse exacte en cas d'erreur. Nous considérons que, donnant cinq bonnes réponses à la suite, le sujet a appris à différencier les deux pianistes. Ainsi, au sortir de cette étape, les sujets ne confondent plus le style Borron et le style Magnien.
   Nous pouvons alors introduire dans l'expérience les simulations sorties de notre algorithme.

2. Phase 2 : phase de test
   Deux types de données sont attendues avec cette phase de test : les données quantitatives, c'est-à-dire le temps mis pour démasquer l'algorithme, et les données qualitatives qui offrent l'opportunité de déterminer ce qui a permis au sujet de trouver l'algorithme.
   Dans la cabine d'écoute, le sujet est devant l'écran sur lequel se trouve une grille contenant 4 cases : Borron, Magnien, Simulation Borron et Simulation Magnien. Pour faire son choix, un joystick. A des intervalles de temps réguliers et très serrés, on relève la position du curseur. Lorsque le sujet est sûr de sa décision, il appuie sur le bouton du joystick et la musique s'arrête, en même temps que le relevé de position. Ce relevé est enregistré dans un fichier excel et on peut alors retracer le parcours précisement.

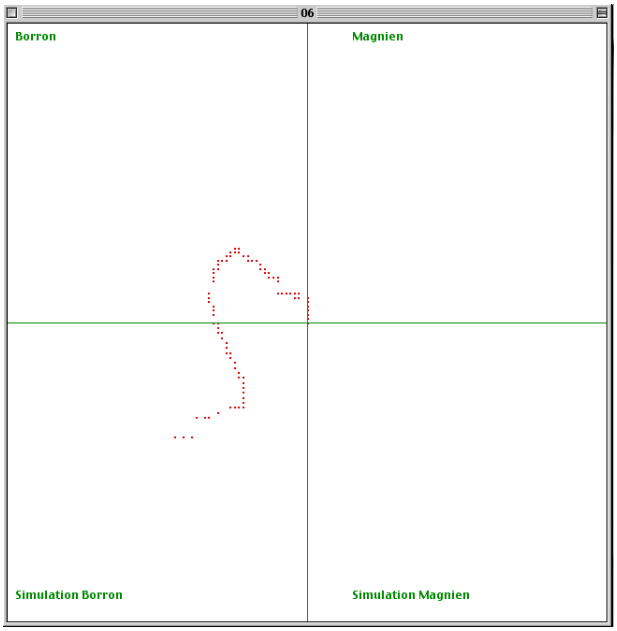


FIG. 25 – Exemple de chemin parcouru par un sujet avant de prendre sa décision

On fait entendre 24 extraits. Le sujet sait qu'il y a des extraits des deux pianistes et des simulations basées sur les deux pianistes également. Par contre, il ne sait pas combien d'extraits de chaque il y a. Nous avons décidé d'en mettre 6 par catégories.

FIG. 26 – Dispositif expérimental

## V.4 Analyse des expériences

Nous avons donc 24 relevés à analyser pour chaque sujet. Pour notre étude, nous avons eu la participation de 12 sujets, dont 11 ont été retenus. Nous allons regarder les résultats, suivant différents critères d'analyse.

### V.4.1 Bonnes réponses

Après la phase d'apprentissage, on sait que les sujets différencient les deux pianistes. Nous voulons savoir s'ils différencient également un pianiste d'une simulation, et si l'identité stylistique se retrouve dans les simulations.

1. Le pourcentage de bonnes réponses
   Voici le tableau des résultats, associant à chaque sujet le nombre de bonnes réponses.

| **SUJET** | **BR /24** | **SUJET** | **BR /24** |
|---|---|---|---|
| Sujet 1 | 13 | Sujet 7 | 15 |
| Sujet 2 | 9 | Sujet 8 | 10 |
| Sujet 3 | 11 | Sujet 9 | 17 |
| Sujet 4 | 14 | Sujet 10 | 14 |
| Sujet 5 | 13 | Sujet 11 | 15 |
| Sujet 6 | 19 | **Moyenne** | **13** |

FIG. 27 – Table des bonnes réponses

On voit ici que la moyenne est de 13 bonnes réponses sur 24 soit un pourcentage de 54%. Le point le plus important au début de l'étude est de savoir si notre expérience n'était ni trop facile, ni trop compliquée. Or, si l'on compare notre premier résultat de 54% de bonnes réponses, on voit qu'il est bien supérieur au hasard, qui est de 25%. C'est donc que quelque chose se passe dans la situation expérimentale. Voyons dans le tableau suivant, p.49, les résultats par catégorie.

| Sujet | Borron | Magnien | Simulation Borron | Simulation Magnien |
|---|---|---|---|---|
| Sujet 1 | 4 | 3 | 3 | 3 |
| Sujet 2 | 3 | 2 | 2 | 2 |
| Sujet 3 | 2 | 4 | 3 | 2 |
| Sujet 4 | 2 | 4 | 3 | 5 |
| Sujet 5 | 4 | 3 | 3 | 3 |
| Sujet 6 | 6 | 4 | 5 | 4 |
| Sujet 7 | 5 | 3 | 3 | 4 |
| Sujet 8 | 3 | 1 | 3 | 4 |
| Sujet 9 | 5 | 4 | 4 | 4 |
| Sujet 10 | 3 | 4 | 5 | 2 |
| Sujet 11 | 5 | 4 | 1 | 5 |
| **Moyenne** | **3.82** | **3.27** | **3.18** | **3.36** |

FIG. 28 – Tableau des bonnes réponses par catégorie. Rappelons qu'il y avait 6 extraits pour chaque catégorie.

Cela donne 64% de bonnes réponses pour Borron, 54.5% pour Magnien, 53% pour le clone de Borron et 56% pour celui de Magnien. Tous ces résultats sont bien au dessus du hasard, ce qui prouve que les sujets ne répondent pas au hasard, ni pour les clones, ni pour les pianistes.
Ceci donne un premier aperçu des résultats. Toutefois, l'analyse des bonnes réponses ne permet pas de distinguer la sensibilité perceptive du sujet de sa stratégie de réponse (cf Théorie de la détection du Signal). Pour différencier ces deux choses, on fait une étude de la sensibilité du sujet.

2. La sensibilité
Dans cette étude, on comptabilise par type d'extraits les hits, c'est-à-dire les bonnes réponses et les fausses alertes (FA), soit le nombre de fois qu'un type a été donné de manière erronée. Par exemple, un sujet trouve 3 fois à raison Magnien, mais choisit ce même pianiste pour 4 autres réponses (donc fausses). On dira que ce sujet n'est pas sensible au style de Magnien car sa sensibilité est de 3 - 4 = -1. Voici le tableau qui résume les résultats en terme de sensibilité.

**hit - FA = Sensibilité**

| Sujet | Borron | Magnien | Simulation Borron | Simulation Magnien |
|---|---|---|---|---|
| Sujet 1 | 1 | 3 | -2 | 0 |
| Sujet 2 | 0 | -2 | -3 | -1 |
| Sujet 3 | -1 | 4 | -2 | -3 |
| Sujet 4 | 1 | 3 | 1 | -1 |
| Sujet 5 | 2 | -1 | 0 | 1 |
| Sujet 6 | 4 | 4 | 3 | 3 |
| Sujet 7 | 3 | 2 | 1 | 0 |
| Sujet 8 | -2 | 1 | -2 | -1 |
| Sujet 9 | 3 | 3 | 2 | 2 |
| Sujet 10 | -2 | 3 | 2 | 1 |
| Sujet 11 | 5 | 2 | -1 | 0 |
| **Moyenne** | **1.27** | **2** | **-0.09** | **0.09** |

FIG. 29 – Tableau d'étude de la sensibilité sur un groupe de 11 sujets

C'est à Magnien, avec une moyenne de 2, que les sujets sont le plus sensible. Cela signifie que les extraits de ce pianiste lui ont été attribués pendant les expériences mais aussi qu'on ne l'associe pas souvent à un autre extrait. Pour le deuxième pianiste, bien qu'un peu moins prononcé, il est également reconnu. Cela, suite à la phase d'apprentissage, confirme que chaque musicien a une identité stylistique.
Si l'on regarde les "clones", les résultats ne sont pas aussi satisfaisants. En effet, une moyenne qui tourne autour de zéro en sensibilité montre que les bonnes réponses étaient équilibrées avec les mauvaises pour chaque simulation, donc, que rien en particulier n'a permis de les discerner des autres.
Le constat, ici, est que la sensibilité des sujets au style des pianistes est supérieure à la sensibilité au style des clones. Deux possibilités : soit on confond les clones avec les humains, ce qui serait flatteur pour l'algorithme, soit on confond les styles des clones entre eux. On ne peut malheureusement pas répondre à cette question avec l'étude de la sensibilité. Nous devons donc faire l'étude des confusions.

### V.4.2 L'analyse des confusions

On cherche à comprendre pourquoi il y a eu des erreurs et dans quel sens elles vont. Pour cela, on s'occupe des mauvaises réponses et pour chaque musicien ou clone, on repère quelles confusions ont été faites. Par exemple, on regarde les résultats des items dont la réponse était Borron et on regarde combien de fois il a été confondu avec Magnien, avec le clone de Magnien

ou avec son propre clone. Le dernier cas serait d'ailleurs très flatteur pour l'algorithme. Nous présentons ci-dessous le tableau des résultats. Attention : B->M se lirait, lorsque j'avais un extrait de Borron, combien ont répondu Magnien.

| **Sujet** | **B / M** | **clB/clM** | **x->cl x** | **x->cl y** | **cl x->x** | **cl x->y** |
|---|---|---|---|---|---|---|
| Sujet 1 | 3 | 6 | 2 | 0 | 0 | 0 |
| Sujet 2 | 3 | 4 | 2 | 2 | 2 | 2 |
| Sujet 3 | 0 | 4 | 3 | 3 | 1 | 2 |
| Sujet 4 | 2 | 4 | 2 | 2 | 0 | 0 |
| Sujet 5 | 3 | 3 | 1 | 1 | 2 | 1 |
| Sujet 6 | 2 | 3 | 0 | 0 | 0 | 0 |
| Sujet 7 | 3 | 5 | 1 | 0 | 0 | 0 |
| Sujet 8 | 2 | 3 | 4 | 2 | 2 | 1 |
| Sujet 9 | 3 | 4 | 0 | 0 | 0 | 0 |
| Sujet 10 | 3 | 2 | 2 | 0 | 0 | 3 |
| Sujet 11 | 0 | 4 | 2 | 1 | 1 | 1 |
| **Total** | **24** | **42** | **19** | **11** | **8** | **10** |
| **Moyenne** | **2.18** | **3.82** | **1.73** | **1** | **0.73** | **0.91** |

FIG. 30 – Analyse des confusions

Ces résultats ne sont pas très satisfaisants pour nous. En effet, premièrement, si l'on s'occupe uniquement des erreurs, soit 45.8% de l'ensemble des réponses, on trouve 21% de confusions entre les deux pianistes.
On trouve ici 36% des erreurs sont dues à une confusion entre le clone de Borron et celui de Magnien. Cela confirme le résultat que l'algorithme perd l'identité stylistique des pianistes.
Les sujets n'ont pas de difficulté à distinguer les clones des humains (15.8% seulement de confusions) mais énormément par contre pour distinguer les styles des clones avec 36% des erreurs, ce qui est nettement plus que les confusions des styles des pianistes (21%).
Ce constat ne semble pas très positif pour l'algorithme. Cependant, un critère important reste à étudier : le temps mis par les sujets pour prendre leur décision.

### V.4.3 Temps

Le temps est un critère important dans nos expériences. En effet, on a vu que peu de clones étaient confondus avec les originaux. Cependant, ce constat pourrait être atténué ou aggravé selon le temps de décision. Si les

sujets remarquent dans les premières secondes qu'il s'agit d'une simulation (d'un clone), c'est qu'elle est trop différente d'un pianiste donc, peu crédible. Au contraire, s'ils mettent plus de temps, c'est que la différence n'est pas claire et que l'algorithme fait illusion, ce qui serait une très bonne chose. Qui plus est, si ce temps d'identification continue à être long en fin d'expérience, cela signifiera que l'algorithme continue à faire illusion après un long temps d'écoute. Pour répondre à ces questions, nous avons calculé le temps qui est mis en moyenne pour découvrir un pianiste et celui pour découvrir un clone. On trouve 35.84s pour le premier et 29.34s pour le second. Le sujet met donc quasiment autant de temps à découvrir le pianiste que le clone. Etant donné la valeur des écart-types (14.68s pour les pianistes et 11.5s pour les clones), ces différences ne sont pas significatives sur un plan statistique.
Ci-dessous le tableau des résultats de l'analyse sur le temps de réponse. On ne comptabilise que les temps des bonnes réponses.

| **Sujet** | **Tps en s** | **Sujet** | **Tps en s** |
|---|---|---|---|
| Sujet 1 | 27 | Sujet 7 | 22.7 |
| Sujet 2 | 36 | Sujet 8 | 34.5 |
| Sujet 3 | 20.5 | Sujet 9 | 31 |
| Sujet 4 | 29 | Sujet 10 | 51 |
| Sujet 5 | 42.5 | Sujet 11 | 35 |
| Sujet 6 | 22.6 | **Moyenne** | **32** |

FIG. 31 – Temps de décision en moyenne par sujet

On voit ici que les décisions ne sont pas rapides car plus de 30 secondes en moyenne est très élevé, d'autant que la consigne expérimentale demandait expressément aux sujets de répondre aussi rapidement que possible. Pour être sur que ces hésitations ne sont pas toujours dans le même sens (choix entre les deux pianistes par exemple), on moyenne sur chaque groupe d'extraits. Voici ce que l'on trouve :

| **Groupe d'extraits** | **Tps en s** | **Ecart-type** |
|---|---|---|
| Borron | 39 | 12.13 |
| Magnien | 32.5 | 17.2 |
| Simul.Borron | 29.5 | 12.2 |
| Simul.Magnien | 30.5 | 9.1 |

FIG. 32 – Temps de réponse en moyenne

En regardant les temps moyens de bonnes réponses associés aux discriminations des deux styles jazzistiques dans les versions pianistes et clones, les

temps montrent que la différence de style est plus rapide à percevoir pour les clones que pour les hommes. Le temps de réponse nous apporte un éclairage nouveau sur la difficulté à différencier les styles chez les pianistes et chez les clones.
Les sujets mettent donc plus de trente secondes pour différencier les deux pianistes, ce qui est assez élevé. Cette durée, notamment avec Borron, peut venir de l'hésitation clone/pianiste. On peut le voir en suivant certains chemins dessinés avec le joystick (cf fig.24). Cependant, on ne peut pas comparer cela chez tous les sujets car pour beaucoup, ils ont attendu d'avoir pris une décision avant de bouger le joystick. On ne peut donc pas généraliser ce résultat. Par contre, ce qui est très intéressant, c'est le fait que les sujets aient mis en moyenne trente secondes pour reconnaitre une simulation. Cela signifie que pendant ces trente secondes, il n'ont pas entendu quelque chose de catégoriquement négatif dans les simulations. C'est un très bon point pour l'algorithme : il fait illusion pendant trente secondes. On peut dire que l'algorithme résiste au temps, modérément bien entendu. Cela rend le constat très positif.

### V.4.4 Item

La dernière étude porte sur les items ou extraits. On cherche à savoir si certains items sont plus "faciles" à reconnaitre que d'autres et ce qui est susceptible de faire prendre une décision. On résume donc dans un tableau le nombre de bonnes réponses et le temps de décision pour chaque extrait d'un clone.

| Item | nbre BR | Tps en s | Item | nbre BR | Tps en s |
|---|---|---|---|---|---|
| 3 - clB | 7 | 39.5 | 15 - clM | 7 | 35 |
| 6 - clM | 10 | 30.3 | 16 - clB | 9 | 35.8 |
| 7 - clB | 5 | 30.7 | 18 - clB | 7 | 25.85 |
| 8 - clM | 7 | 28.65 | 20 - clM | 6 | 28.6 |
| 9 - clM | 4 | 41.54 | 21 - clM | 2 | 25.8 |
| 11 - clM | 4 | 33.4 | 24 - clB | 1 | 26.5 |

FIG. 33 – Temps de réponse en moyenne, en secondes. (B=Borron, M=Magnien, clB=clone Borron,clM=clone Magnien)

En général, les résultats sont assez équilibrés. Ceci est un résultat important : cela signifie que l'algorithme travaille de façon constante, homogène.
On repère tout de même quelques difficultés avec les extraits 21 et 24. Mais, cela ne s'explique pas par le phénomène de lassitude, de déconcentration

qu'aurait pu subir le sujet puisqu'aucun n'a entendu les extraits dans le même ordre. Donc, les extraits 21 et 24 n'étaient pas forcément les derniers. Regardons de plus près l'extrait n°6 : 10 personnes ont donné la bonne réponse. L'ecart-type des temps de réponse de ces 10 sujets est de 14.7. Il n'y a donc pas un élément radical au choix pour cet extrait. On regarde l'extrait 16 : l'ecart-type est de 15, ce qui signifie qu'il y a une grande variation entre les sujets.
On peut donc dire que l'algorithme fonctionne de façon homogène.

### V.4.5 Comparaison des deux algorithmes

Dans les 12 extraits des clones (6 Magnien, 6 Borron), il y avait pour chaque catégorie la moitié générée avec la fonction de comparaison Estrada et l'autre celle des classes d'accords. On veut comparer les deux pour savoir laquelle est la plus efficace.

1. Les bonnes réponses On a un pourcentage de bonnes réponses de 57.6% sur les clones provenant d'Estrada contre 51.5 pour la table d'accords. Pour le clone de Borron, les résultats sont équilibrés. Pour le clone de Magnien, c'est Estrada qui fonctione le mieux, repéré à 63% contre 48.5 seulement pour la table d'accords.
2. Le temps de réponse En moyenne, les sujets mettent 33.8s pour découvrir un "extrait Estrada" contre 29.8s pour la table d'accords. Si l'on détaille, voici les résultats :

| Comparaison | pianiste | Tps en s |
|---|---|---|
| Estrada | Borron | 32 |
| Estrada | Magnien | 35.6 |
| Tableau | Borron | 29.4 |
| Tableau | Magnien | 30.22 |

FIG. 34 – Comparaison Estrada/Table d'accords

On voit dans ce tableau, que Estrada capture mieux le style, en particulier celui de Magnien.

## V.5 Discussion et perspectives

### V.5.1 Discussion

Explicitons un peu mieux ce que suggèrent ces résultats.
Le premier bilan est assez positif puisque, certes, les sujets reconnaissent souvent les clones mais ils mettent du temps pour cela. L'algorithme ferait

illusion plusieurs dizaines de secondes, ce qui est loin d'être mauvais. De plus, le fait qu'il fonctionne de façon homogène est une qualité.
Par contre, l'algorithme peut être critiqué sur un point important : il y a une perte fondamentale dans la capture des styles. Ils ne sont pas modélisés, comme nous l'espérions puisque les sujets n'y sont plus sensibles, et au contraire, on perd l'opposition stylistique entre les deux pianistes. C'est un point sur lequel il serait bon de retravailler.
Lors de l'analyse des résultats, nous avons fait un constat surprenant, avec un nombre non négligeable d'extraits originaux qui étaient pris pour des clones. L'explication viendrait du fait que, pour enlever un biais certain, tous les extraits ont été joués en midi. Cela a pu gêner certains sujets car la musicalité en est amoindrie, ce qui pourrait influencer à choisir un clone. Cependant, la ligne mélodique y reste présente, alors qu'elle n'est pas sensée y être dans une simulation (pas de contrainte mélodique). De plus, signalons, que les simulations sont, de part leur passage dans l'algorithme, plus quantifiées que les extraits originaux. Cela prouve que l'algorithme n'a pas été favorisé par l'expérience.
Un autre point est que la stratégie de réponse des sujets était influencée par le fait de savoir qu'il y avait des pianistes et des simulations. Alors, dès qu'ils entendaient quelque chose d'un peu étrange, ils se dirigeaient vers les clones. On pourrait recommencer l'expérience mais cette fois, en annonçant au sujet qu'il y a quatre pianistes différents pour la phase deux de l'expérience.
L'oracle des facteurs est donc intéressant, en particulier avec la fonction de comparaison Estrada qui donne de meilleurs résultats. On pourrait continuer à travailler ces méthodes de comparaison car il est maintenant prouvé qu'elles jouent un rôle important.
C'est donc un bilan plutôt positif pour l'algorithme qui fait illusion environ 30 secondes. Rappelons que les critiques de l'algorithme sont pondérées par le fait que les sujets étaient des musiciens et amateurs voire connaisseurs en jazz.

### V.5.2 Perspectives

Pour améliorer les résultats de l'algorithme, nous avons déjà implémenté d'autres fonctions de comparaison utilisées dans l'analyse du fichier. Elle est basée sur la recherche de basse d'Hindemith, décrite en IV.3.5.6. On pourrait également mettre des contraintes sur la vélocité. En effet, on pourrait obliger le choix vers une note qui a une vélocité proche de celle de la note de départ. Cela éviterait des sauts qui nuisent à la souplesse de la phrase musicale.
Quelques points pourraient être améliorer sur l'expérience : le problème du midi serait atténué en faisant jouer les simulations par le disklavier (piano midi) et en les enregistrant en audio. Ainsi, on pourrait utiliser les fichiers

originaux et limiter les confusions homme-machine au sens qui nous intéresse.

L'étape suivante, qui reste à explorer, part du problème constant de logique harmonique, sur laquelle nous n'avons aucun contrôle. Si l'on veut rester dans les modèles statistiques, on pourrait séparer la mélodie (chorus), de la partie accompagnement. Sur cette dernière, on ferait une analyse harmonique et en dériverait une chaîne de labels harmoniques. On aurait donc un nouvel alphabet, constitué des éléments du chant et des labels harmoniques. On se rapprocherait du système "à multiples points de vue" décrit par Conklin et Witten[6].Cela nous amènerait à un niveau de contrôle supérieur.
Dans le cas d'un résultat positif, on pourrait alors produire une situation d'interactivité entre le pianiste et la machine, l'un jouant le thème et l'autre y ajoutant l'harmonie ou inversement. Cela aboutirait à de nouvelles expériences perceptives d'un autre niveau. Il faudrait imaginer un pianiste jouant une des parties, en duo avec soit un autre pianiste, soit une machine, l'objectif étant de savoir si le sujet est capable de reconnaître qui est son partenaire.

# VI Remerciements

Sincères remerciements à :

- Gérard Assayag et Emmanuel Bigand pour leur encadrement, leurs conseils, pour m'avoir permis de mener cette étude dans les meilleures conditions,

- l'équipe Représentations Musicales de l'IRCAM pour son accueil et sa disponibilité,

- l'équipe Perception et Cognition Musicale également pour son accueil durant les expériences en cabine d'écoute,

- LEAD (Laboratoire d'Etude de l'Apprentissage et du Développement) pour le matériel de l'expérience,

- Pascal Borron et Bernard Magnien pour leur sympathique participation aux concerts,

- l'équipe de techniciens qui s'est occupée de les enregistrer,

- l'ensemble des sujets qui nous ont accordé un peu de leur temps pour passer l'expérience.

# VII Bibliographie

## Références


[1] A.Lefebvre and T.Lecroq. Computing repeated factors with a factor oracle. *Proceedings of the 11th Australasian Workshop On Combinatorial Algorithms*, 2000.

[2] Les arbres de recherche. http ://www.cs.mcgill.ca/ cs251/oldcourses /1997/topic7/. 1997.

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

[4] La compression LZ. http ://perso.wanadoo.fr/lebazar/tipe/tipe.htm. 2000.

[5] David Cope. *Computers and musical style.* Oxford University Press, Oxford, 1991.

[6] D.Conklin and I.H.Witten. Multiple viewpoint system for music prediction. *Journal of New Music Research*, 24 :p51–73, 1995.

[7] D.Hankerson, G.A.Harris, and Jr P.D.Johnson. *Introduction to Information theory and data compression.* CRC Press LLC, Boca Raton, Florida, 1998.

[8] D.Long and W.Jia. Optimal maximal prefix coding and huffman coding. *Department of Computer Science*, 2001.

[9] D.Ron, Y.Singer, and N.Tishby. The power of amnesia : Learning probabilistic automata with variable memory length. *Machine Learning*, 1996.

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

[11] F.Pachet. Surprising harmonies. *International Journal on Computig Anticipatory Systems*, 1999.

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

[13] G.Bejerano and G.Yona. Variations on probabilistic suffix trees - a new tool for modeling and prediction of protein families. *Bioinformatique*, 2000.

[14] G.Berejano and G.Yona. Modeling protein families using probabilistic suffix trees. *Proceedings of the 3rd Annual International Conference of Computational Molecular Biology*, 1999.

[15] I.Bent and W.Drabkin. *L'analyse musicale.* Main d'oeuvre, 1987.

[16] L.Chéno. Automate des suffixes. application à la recherche d'un motif dans un texte. *Colloque Luminy*, 2001.

[17] Demonstration LZW. http ://www.cs.sfu.ca/cs/cc/365/li/squeeze /lzw.html. *Internet*, 2000.

[18] Thierry Meulemans. *L'apprentissage implicite.* Solal, 1998.

[19] M.Feder, N.Merhav, and M.Gutman. Universal prediction of individual sequences. *Convention of Electrical and Electronics Engineers In Israel*, 17th, 1991.

[20] 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.

[21] O.Lartillot, S.Dubnov, G.Assayag, and G.Bejerano. Automatic modeling of musical style. *Proceedings of Journées de l'Informatique Musicale*, 2001.

[22] O.Lartillot, S.Dubnov, G.Assayag, and G.Bejerano. A system for computer music generation by learning and improvisation in a particular style. *Proceedings of the International Computer Music Conference*, 2001.

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

[24] LZW par Mark Nelson. http ://www.dogma.net/markn/articles/lzw /lzw.htm. *Dr Dobb's Journal*, octobre, 1989.

[25] P.Levy. *La machine univers.* La découverte / Sciences et société, 1987.

[26] R.Cole and R.Hariharan. Faster suffix tree construction with missing suffix links. *Proceedings of the 32nd Annual Symposium on the Theory of Computing*, pages 407–415, 2000.

[27] R.Giegerich and S.Kurtz. From ukkonen to mccreight and weiner : A unifying view of linear-time suffix tree construction. *International Computer Science Institut*, 1997.

[28] Charles rosen. *Le style classique.* tel-Gallimard, 2000.

[29] S.Canazza and N.Orio. How are the players ideas perceived by listeners : Analysis of "how the moon" theme. *Proceedings of Kansei-The technology of Emotions, AIMI, International Workshop, Genova*, 1997.

[30] S.Canazza, G.De Poli, and A.Vidolin. Perceptual analysis of the musical expressive intention in a clarinet performance. *M.Leman (ed), Music, gestalt, and computing*, 1997.

[31] S.W.Golomb, R.E.Peile, and R.A.Scholtz. *Basic concepts in information theory and coding.* Plenum, 1994.

[32] T. H. Wonnacott and R. J. Wonnacott. *Introductory Statistics for Business and Economics, fourth edition.* John Wiley and Sons, New York, 1990.