Projet Sample Orchestrator 2
D4.3.3 - Hybrid Model V01

Version	Date	Contributeurs	Validation	Contenu
V01	Avril 2012	Laurent Bonnasse-Gahot (principal chercheur et développeur, rédaction), Benjamin Lévy, Gérard Assayag	Gérard Assayag, Hugues Vinet	Représentations et Modèle d'apprentissage et de génération hybride avec modèle temporel issu du corpus

Résumé:

L'objet de ce document est de présenter l'architecture et le fonctionnement du nouveau prototype de logiciel d'harmonisation/arrangement à la volée. Un tour d'horizon de la dernière version est également proposé à travers une description de son interface.

Ircam, Équipe Représentation Musicale
SOR2 – WP4

Prototype de logiciel d'harmonisation/arrangement à la volée : SOMax v0.1

Laurent Bonnasse-Gahot

Mars 2012

Rapport interne

Table des matières

1 Introduction

Ce rapport a pour but de présenter mon travail effectué dans le cadre du projet Sample Orchestrator 2, d'octobre 2011 à mars 2012 (à temps partiel 3/5), en collaboration avec Gérard Assayag, Benjamin Lévy, Georges Bloch, Carine Bonnefoy, Moritz Reich et Remy Muller.

1.1 Cadre et objectifs du présent projet

Le but du projet est d'explorer de nouvelles possibilités d'interactions musicales entre un musicien et la machine en proposant d'harmoniser/arranger en temps réel le jeu d'un musicien. Il s'agit pour cela de mettre en relation le jeu du musicien (également appelé utilisateur par la suite) avec une mémoire musicale (corpus) en accordant au mieux leurs logiques harmonique (verticale), mélodique (horizontale) et temporelle (tempo, pulsation, voire métrique). En fonction de la fonction d'appariement concernée, on parlera d'harmonisation de mélodie, d'arrangement, d'accompagnement, ou encore de rendu musical augmenté. Une contrainte forte du projet est d'imposer à la machine de réagir en temps réel, ce qui implique un cadre causal. De plus, il s'agit de traiter une entrée polyphonique tout autant que monophonique.

1.2 Quelques précisions et mises en garde

Ce rapport ne vise pas à rendre compte de façon exhaustive des différentes idées explorées ni des détails d'implémentation. Seules les grandes lignes sont présentées. Il ne s'agit pas d'un article, même si un article pourra à l'avenir se baser sur du matériel présenté ici, d'où notamment une bibliographie particulièrement succincte et une absence de mise en perspective sur le travail effectué.
Au niveau du logiciel proposé, il faut noter qu'il s'agit là d'une version prototypique qui, si elle permet déjà d'expérimenter avec différents modes de jeu, présente de nombreuses limites. Elle ne sera pas maintenue à l'avenir, en raison des changements importants qui vont rapidement être opérés au cœur du système.

2 Principe général de fonctionnement

2.1 Aperçu

Le présent projet se base sur le projet OMax[1] (Assayag *et al.*, 2006), né des travaux sur la simulation stylistique menés par l'équipe Représentation Musicale à l'Ircam (Dubnov et Assayag, 1998; Assayag *et al.*, 1999; Poirson, 2002; Assayag et Dubnov, 2004). Le logiciel OMax construit un modèle du jeu du musicien au fur et à mesure qu'il le capte (grâce à l'algorithme d'oracle des facteurs introduit par Allauzen *et al.*, 1999). Il est ensuite possible de naviguer à l'intérieur de cette représentation en empruntant des chemins différents de celui pris par le musicien, ce qui conduit à générer des improvisations originales dans l'esprit du jeu de l'artiste.
Avec OMax, la navigation dans cette structure se fait de façon essentiellement libre. Il s'agit ici de guider cette navigation en fonction de ce que joue le musicien au moment où il le joue. De plus, si (en règle générale) OMax part d'une mémoire vide qu'il construit au fur et à mesure de son écoute du jeu d'un musicien, on part ici d'une mémoire préétablie.

2.2 Mémoire musicale

Les données musicales que l'on considère correspondent à du matériel MIDI polyphonique. Suivant la logique proposée par OMax, un flux polyphonique MIDI est d'abord transformé en une séquence de *tranches polyphoniques* (une tranche polyphonique correspond à un ensemble de notes, ou "accord", associé à une certaine durée, et constitue l'unité de base de la mémoire musicale).
Cette séquence est ensuite analysée de sorte à détecter les motifs répétés présents dans la séquence. Cette analyse permet d'obtenir un graphe dont les états, correspondant à chacune des tranches polyphoniques du matériel MIDI d'origine, sont reliés s'ils possèdent un suffixe commun (appelé contexte par la suite). Si une lecture de ce graphe d'un état à son état suivant revient à rejouer la séquence d'origine, une lecture autorisant des sauts (passage d'un état à un état différent de l'état suivant) engendre une séquence musicale différente de la séquence d'origine, mais qui respecte son style, sa logique interne.
Le but ici, une fois une telle mémoire musicale constituée, est alors de pouvoir naviguer dans cette mémoire en tenant compte – à la volée – de différentes contraintes, et tout particulièrement du jeu du musicien.

1. `http://omax.ircam.fr/`

2.3 Navigation guidée

Supposons que l'on se trouve à un endroit donné de la mémoire. Un événement [2] demande au système de lui fournir, si possible, un nouvel état partant de l'état courant, pour qu'il soit joué à un temps t. La structure d'oracle nous permet de considérer un ensemble d'états comme suite possible à l'état courant et qui respecte ce que l'on a appelé la logique interne de la mémoire musicale. Cet ensemble d'états est dans un premier temps soumis à une (ou plusieurs) condition(s) stricte(s) (étape de filtre). Les états ne respectant pas cette condition sont supprimés de la liste des états à considérer (avec, par conséquent, la possibilité de n'avoir plus aucun état à considérer, d'où la nécessité d'un mécanisme de gestion des échecs, détaillé en section 4.2.1). Typiquement, une condition stricte utilisée en pratique consiste à regarder si la note actuellement jouée est incluse dans la tranche (cette inclusion pouvant, comme nous le verrons, revêtir des aspects différents, et induire un rendu différent). L'étape suivante consiste à évaluer les solutions obtenues en fonction de différents critères, que l'on peut voir comme des conditions plus souples. Ceci permet de favoriser une solution par rapport à une autre, par exemple parce que le saut correspondant est de meilleure qualité (au sens où le contexte est plus long). Cette étape d'évaluation est alors suivie d'une étape de sélection, où une solution en particulier est choisie de façon aléatoire, en fonction justement de l'évaluation. L'élu est alors envoyé pour être joué au temps t demandé initialement, ou à un temps corrigé pour satisfaire une condition rythmique.

En résumé, la recherche d'un nouvel état s'effectue suivant la chaîne de traitement suivante :
ensemble de sauts possibles → filtre → évaluation des solutions → sélection d'une solution → choix d'une date appropriée → jeu de cette solution

En fonction de l'événement qui déclenche la recherche d'un nouvel état dans la mémoire, le comportement du logiciel diffère significativement. Lorsqu'il s'agit de la détection d'une nouvelle note jouée par le musicien, le logiciel fonctionne en mode 'note à note', et joue en même temps que le musicien. À l'opposé, lorsque l'événement déclencheur est purement interne, c'est-à-dire qu'il ne dépend pas de l'entrée utilisateur, la machine peut générer de façon autonome de la musique, suivant le style du corpus utilisé. La combinaison de ces deux événements, couplée au choix de la méthode d'appariement, permet d'obtenir une variété d'interactions et de rendus (harmonisation, augmenta-

2. Nous verrons plus loin que l'on considère en pratique deux types d'événements : la fin d'une tranche ou la détection d'une nouvelle entrée utilisateur.

tion, arrangement, orchestration), comme on le verra un peu plus en détails en section 4.

3 Corpus

3.1 Constitution

Un corpus est constitué à partir d'un ensemble de fichiers au format MIDI. L'analyse de ce corpus s'effectue suivant la logique développée dans le logiciel OMax, que nous rappelons brièvement ici.
Dans un premier temps, il s'agit de segmenter le matériel polyphonique considéré en éléments atomiques appelés *tranches polyphoniques* (voir Dubnov *et al.*, 2003), chaque nouvelle tranche correspondant au début d'une nouvelle note ou d'un ensemble de notes (en tenant compte des légers décalages temporels éventuels inhérents au jeu d'un accord). Cette opération permet de transformer le flux initial en une séquence de tranches. L'étape suivante consiste alors à analyser cette séquence, dans le but d'en extraire les répétitions de motifs. Cette analyse est effectuée grâce à l'algorithme d'oracle des facteurs développé par (Allauzen *et al.*, 1999) (en tenant compte des améliorations algorithmiques introduites par Lefebvre *et al.*, 2002).
Il est important de noter qu'en pratique, cette analyse s'effectue non pas directement sur le contenu des états (tranches), mais sur une séquence de symboles associés à ces états. En effet, il est nécessaire d'introduire une fonction de similarité qui permet de dire, au niveau de l'analyse, si deux unités sont équivalentes. Si l'on peut en théorie vouloir travailler sur le contenu harmonique brut, ce qui revient à considérer que chaque ensemble de notes définit sa propre classe d'équivalence, ceci n'a que peu d'intérêt musical puisqu'alors très peu de motifs (au niveau de la séquence) seraient alors trouvés, d'où une inventivité faible. Catégoriser les tranches en un nombre plus restreint de classes permet de repérer des sous-séquences communes, qui, si elles diffèrent du point de vue de leur contenu exact, ont tout de même une forme de similarité qui permettra de sauter de l'une à l'autre de façon cohérente et intéressante sur le plan musical. Plus le nombre de classes est faible, plus il existe a priori de chances de détecter des sous-séquences communes suffisamment longues. Mais trop compresser peut conduire à identifier comme équivalentes des tranches au contenu trop différent. À noter que, du coup, la notion de qualité d'un saut (au sens de la longueur du contexte commun) dépend évidemment de la notion d'équivalence utilisée.
Pour le moment, deux fonctions de classification sont proposées. La première, conçue par Georges Bloch, appelée `virfun gb` dans le logiciel, reprend la si-

milarité utilisée dans le mode MIDI d'OMax. Elle consiste à représenter une tranche par la classe de hauteurs de sa fondamentale virtuelle (ou sa fondamentale d'Hindemith, si elle existe), soit 12 catégories possibles. La deuxième considère le couple (fondamentale, basse) – fondamentale au sens *virfun gb* –, chaque élément étant compris en termes de classe de hauteurs, *ie* modulo 12, soit au total 144 classes. En pratique, de nombreuses combinaisons (fondamentale, basse) ne sont pas rencontrées, soit un nombre de classes effectif plus faible.
Cette notion d'équivalence est importante, et sera explorée plus profondément à l'avenir, à la fois par une évaluation des fonctions utilisées maintenant et la recherche de nouvelles fonctions de classification.

Un outil de création de corpus a été implémenté en MATLAB, de sorte à pouvoir générer des gros corpus rapidement. S'il est possible de créer des corpus depuis OMax, le développement d'un outil de création hors-ligne s'est rapidement imposé. La création dans OMax se faisant en temps réel, le processus devient rapidement laborieux dès que le corpus est un peu long et/ou qu'il est constitué de plusieurs fichiers. L'outil MATLAB développé permet de générer très rapidement un corpus à partir d'un ensemble de fichiers MIDI. De plus, les récents développements du logiciel ont demandé l'utilisation des données rythmiques de position par rapport à la pulsation et de tempo que ne peut gérer OMax, d'où le besoin de développer un outil propre au présent projet. Enfin, cet outil permet de tester différentes fonctions de classification, ce que ne permet pas la version courante d'OMax. À noter que de nouveaux objets C, développés en collaboration avec Benjamin Lévy (voir section 6.2) vont bientôt être intégrés au projet, et permettront de gérer ces deux derniers points directement dans le logiciel. À terme, on pourra donc si on le souhaite créer des corpus à la volée (tout en continuant à bénéficier évidemment des corpus précalculés).

3.2 Analyse des classes d'équivalence

L'outil MATLAB développé permet également d'analyser, pour un corpus donné, chaque classe d'équivalence, ce qui est un moyen, complémentaire de l'évaluation subjective par l'écoute et l'expérimentation, d'évaluer la pertinence de la fonction de classification utilisée [3]. Étant donnée une fonction de classification, cette analyse permet de visualiser les représentants de chaque

3. Inversement, pour une fonction de classification bien définie, ceci permet également une analyse statistique des motifs verticaux (accords) les plus fréquents dans une œuvre donnée, chez un compositeur, etc.

classe en fonction de leur fréquence d'apparition, chaque membre correspondant à un motif vertical unique, indépendamment de sa durée.
La Figure 1 reproduit un extrait du fichier pdf automatiquement généré par l'outil d'analyse développé. Elle présente les dix représentants les plus fréquents de la classe C (do), suivant une analyse basée sur virfun gb (voir section 3.1).

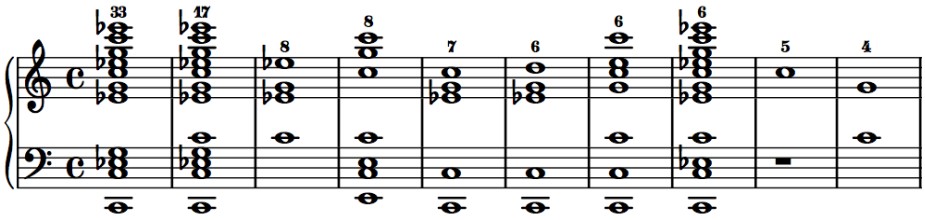

FIGURE 1 – Un extrait d'analyse des motifs verticaux les plus fréquents appartenant à la classe C dans le premier mouvement de la cinquième symphonie de Beethoven.

4 Navigation guidée

On rappelle ici la chaîne de traitement présentée en section 2.3, en décrivant chaque partie plus en détails.
La combinaison des paramètres utilisés en chaque point de cette chaîne et du choix du type d'événement entraînant la recherche d'un nouvel état offre d'ores et déjà une certaine richesse d'interaction. Plusieurs 'méta-modes', correspondant à une configuration particulière de ces paramètres, sont aujourd'hui proposés, comme évoqué en section 2.3. Ils sont décrits plus explicitement dans cette section.

4.1 Chaîne de traitement

4.1.1 Ensemble de sauts possibles

À partir d'un état donné, on désire accéder à l'ensemble des continuations possibles. Ceci est donné grâce au calcul de l'arbre des liens suffixiels (Suffix Link Tree, SLT par la suite) à partir de la structure d'oracle (voir Assayag et Bloch, 2007). Cet arbre est élagué de sorte à ne conserver que les liens vers des suites possibles au-dessus d'une certaine qualité. La qualité correspond ici à la longueur du contexte, paramètre important du système. Plus le

contexte est long, plus les sauts sont de bonne qualité, au sens où ils suivent mieux la logique interne du corpus (du point de vue, il est bon de le rappeler, de la fonction d'équivalence utilisée), mais moins ils sont nombreux. Si d'un côté l'on souhaite avoir des sauts de bonne qualité, imposer un contexte commun trop grand entraîne inévitablement une diminution du nombre de sauts possibles, ce qui peut engendrer un appauvrissement de la musique générée, moins de surprise. Plus important pour notre projet, moins l'on a d'états à disposition, plus on a de chances de ne pas avoir de solutions après l'étape de filtre. Cela crée une situation d'échec, au sens où la contrainte forte imposée au système, typiquement un appariement entre l'entrée utilisateur (notes jouées par le musicien) et le contenu musical de l'état, n'est pas satisfaite (voir section 4.2.1).

4.1.2 Filtre

Cette étape consiste à ne conserver que les états qui satisfont une contrainte stricte, pertinente pour le rendu musical recherché. Cette contrainte pourrait par exemple être rythmique, harmonique, mélodique, ou encore porter sur les nuances. La version actuelle du logiciel considère l'inclusion du point de vue de la hauteur tonale. Pour une entrée donnée, un état est conservé (constitue une solution) si toutes les notes composant l'entrée sont incluses dans l'état. Plusieurs modes d'inclusion sont possibles. Il peut s'agir d'un appariement strict, pour lequel chaque note doit exister à son emplacement d'origine (exemple : si l'utilisateur joue un do médian, on recherche la présence d'un do médian dans la tranche). Cet appariement peut également se faire modulo 12, c'est-à-dire sur les classes de hauteurs, chaque classe regroupant toutes les notes de même nom. Dans l'exemple précédent, un état est une solution s'il contient un do, où qu'il soit situé dans l'échelle des hauteurs. Deux contraintes supplémentaires sont également considérées. Le mode soprano impose que la voix la plus aiguë de l'entrée utilisateur corresponde également à la voix la plus aiguë dans l'état, et ce de façon absolue ou en équivalence d'octave (modulo 12). De même, le mode basse impose un appariement équivalent pour la voix la plus basse. À noter que ces modes peuvent être combinés. Au final, le choix du mode d'appariement entre l'entrée utilisateur et la mémoire musicale de la machine est déterminante pour le rendu musical recherché. D'autres modes d'appariement seront éventuellement étudiés et implémentés à l'avenir.

4.1.3 Évaluation des solutions

L'étape suivante consiste à évaluer chacune des solutions en fonction de critères plus souples que l'étape de filtre. Par construction, ces solutions respectent la logique de la mémoire musicale (voir section 4.1.1) et satisfont la contrainte d'appariement avec l'entrée. Si toutes ces solutions sont désormais des candidats potentiels à être joués, elles ne sont pas forcément toutes de même qualité. Cette étape permet de favoriser certaines solutions plutôt que d'autres. Plusieurs critères sont actuellement considérés.

État suivant et longueur de contexte. Un point important porte sur l'évaluation de l'état suivant par rapport aux autres états (sauts dans la mémoire). Mettre un poids très fort sur l'état suivant conduira à jouer systématiquement cet état, ce qui revient à rejouer la séquence d'origine (sous réserve, évidemment, que la contrainte d'appariement soit respectée). Inversement, un poids très faible associé à cet état implique un saut systématique, soit a priori plus de nouveauté par rapport à la séquence d'origine. Dans le cas où l'on saute d'un endroit à un autre de la mémoire, on peut vouloir privilégier les sauts qui possèdent un contexte commun le plus grand possible. Ceci est au cœur même de la logique d'OMax, et permet d'engendrer de la nouveauté tout en respectant l'esprit du matériel musical d'origine. Cependant, le matériel utilisé correspond à des fichiers MIDI d'œuvres contenant des sections entières dupliquées (en raison de la présence de barres de reprise). Ces copies entraînent la présence dans la séquence analysée de contextes de longueur très grande. Favoriser les contextes les plus grands peut alors conduire à la présence de sauts qui n'apportent pas de nouveauté par rapport au fait de suivre la séquence d'origine, ce que l'on ne veut pas si l'on cherche, justement, à sauter. D'où le choix d'une fonction de notation de longueur de contexte en forme de V inversé visant à privilégier les sauts de contexte long mais pas trop. Cette fonction a un impact important sur le rendu musical, et peut être entièrement paramétrée par l'utilisateur.

Position par rapport à la pulsation. Un autre critère d'évaluation regarde la position de la tranche à jouer par rapport à la pulsation (phase). Il s'agit de favoriser les états dont la phase est proche du moment où l'on recherche un état. Par exemple, si l'on recherche un état sur le contretemps, on peut vouloir favoriser les états initialement placés sur un contretemps. Ceci peut être intéressant dans le cas où l'on veut tenir compte, ou tirer profit d'interactions rythme/harmonie potentielles. Plus important dans le

cadre du fonctionnement actuel de notre logiciel, dans le cas où l'on désire jouer l'état sélectionné à son emplacement (phase) d'origine (voir section 4.1.5), favoriser les états dont la phase correspond au moment où l'on désire jouer l'état permet de garder une cohérence rythmique forte par rapport à la séquence d'origine, en synchronisation avec l'horloge interne du logiciel (transport). La même idée sera bientôt implémentée (les objets OMax utilisés actuellement ne le permettent pas, voir section 6.2) à une échelle plus grande qu'une pulsation, de manière à introduire des éléments de métrique (par exemple en tenant compte de la notion de temps fort vs temps faible).

Tempo. Toujours dans les considérations temporelles, l'utilisateur peut favoriser les solutions dont le tempo d'origine est proche du tempo global choisi. Les états peuvent en effet être rejoués suivant un tempo maître défini par l'utilisateur. Un écart trop important entre le tempo imposé et celui d'origine peut entraîner le jeu de tranches à la durée effective trop petite ou trop grande par rapport au résultat attendu, d'où cette possibilité de favoriser les solutions dont le tempo d'origine est proche du tempo global choisi.

Passés communs. Bien que traité de façon relativement sommaire pour le moment, ce point est particulièrement important, et est à explorer de façon plus approfondie. L'idée sous-jacente est de privilégier les états qui respectent la logique de l'utilisateur (en plus de respecter, par essence, la logique de la mémoire musicale interne). Par exemple, imaginons que l'utilisateur ait joué la séquence A $\rightarrow$ B $\rightarrow$ C, cette dernière note correspondant à la note en cours. À ce stade de la chaîne de traitement, on considère les états qui à la fois sont des suites possibles à l'état courant et contiennent la note C. L'idée ici est alors, parmi ces états-là, de favoriser ceux dont les états précédents contiennent les notes A et B. Cette comparaison entre le passé du musicien et le passé de l'état se fait note à note et état par état (plus précisément, vu que l'entrée peut être polyphonique, accord par accord et état par état), les états les plus récents jouant un rôle plus important dans l'évaluation. Au final, plus le passé d'un état est en accord avec le passé du jeu du musicien, plus cet état est privilégié, selon un poids contrôlable par l'utilisateur.
Il faut ici remarquer que cette évaluation de la similarité entre le passé de l'utilisateur et le passé d'un état donné n'est pas évidente, en particulier dans le cas où l'on permet au système d'avoir sa propre temporalité. Reprenons l'exemple précédent, et imaginons que la note C est tenue par l'utilisateur : dans ce cas, si le système génère de la musique sous la contrainte d'avoir

toujours un C, favoriser les états dont le passé contiendrait A puis B n'a de sens que pour le premier état généré, mais n'a sans doute pas beaucoup de sens pour les états suivants.
Cette question du lien entre logiques du musicien et de la mémoire est une des questions importantes à explorer à l'avenir.

Tabou. Lorsque la taille du corpus est relativement faible, et/ou pour certaines associations de contraintes/préférences, des boucles peuvent apparaître, au sens où la machine repasse toujours, et de façon cyclique, par les mêmes états (voir Assayag et Bloch, 2007, dans le cas d'OMax). Si cette situation n'est pas nécessairement indésirable, on peut cependant vouloir éviter ce type de cas. De façon plus générale, on peut vouloir éviter de repasser par un état déjà rencontré. Pour cela on conserve trace des n derniers états générés, et l'on décide, s'ils sont de nouveaux candidats potentiels, de plus ou moins les pénaliser en divisant leur probabilité d'être sélectionnés par une valeur plus ou moins grande (contrôlable par l'utilisateur).

À (re)venir. La version précédente de SOMax intégrait deux autres critères, qui seront peut-être prochainement réintégrés. Le premier consiste à favoriser les états qui conservent la même logique d'appariement d'un enchaînement à un autre. Par exemple, considérons un corpus de 4 voix : si l'entrée de l'utilisateur est incluse dans les voix 2 et 3, on peut vouloir conserver cette logique d'un état à l'autre, et favoriser ce même emplacement lors de l'état suivant.
Le deuxième critère vise à favoriser les états qui ont une densité de notes similaire à l'état courant, de sorte à garder une certaine cohérence dans le nombre de voix jouées.
Les explorations sur une éventuelle notion de conduite de voix (voir section 6.2) pourront rendre caducs ces critères.

4.1.4 Sélection

Cette étape consiste à choisir la solution qui sera effectivement jouée au temps demandé. Ce choix est ici simplement effectué de façon aléatoire, chaque état ayant une probabilité d'être choisi proportionnelle à la note qu'il a reçu à l'étape précédente.

4.1.5 Choix d'une date appropriée

Une fois un état choisi, il s'agit de le jouer à un moment approprié. Deux stratégies sont mises en œuvre. La première consiste naturellement à jouer l'état à la date demandée (rappelons que la requête d'un nouvel état s'accompagne toujours d'une date à laquelle il est prévu a priori de jouer cet état). La seconde consiste à jouer l'état à une date à la fois proche de la date demandée et qui respecte la phase d'origine de l'état. Ceci permet non seulement de rester synchronisé avec l'horloge interne (transport) et d'assurer ainsi une certaine cohérence rythmique, mais également de conserver le détail rythmique (*microtiming*) de la séquence d'origine, de sorte à restituer le groove, le phrasé original.
Une autre façon de "corriger" la date à laquelle jouer un état pourrait consister à aligner cette date selon une grille temporelle déterminée (quantification). Si l'on perd ici le détail rythmique évoqué précédemment, on gagne en possibilité de recombinaison, tout en restant synchronisé avec l'horloge interne. Ce point reste à étudier et à implémenter (éventuellement).

4.1.6 Lecture de la tranche

Une fois déterminés l'état à lire et la date à laquelle le lire, il reste à effectivement lire cette tranche. Il s'agit de transformer le contenu de la tranche en informations MIDI, opération inverse à celle décrite en section 3.1, en fonction du contenu en train d'être joué. En bref, une note peut soit être arrêtée, soit être prolongée, soit enfin être commencée. Lorsqu'une note est jouée, trois possibilités sont offertes : (1) utiliser sa vélocité originale (mode par défaut dans le cas génération pure) ; (2) décider que sa vélocité est déterminée par la vélocité du jeu de l'utilisateur, de sorte que le musicien impose sa nuance à l'ensemble du rendu (mode par défaut dans le cas note à note) ; (3) imposer une valeur fixe comme vélocité de chaque note jouée par le système. Les informations de canal présentes dans le matériel d'origine sont préservées. Enfin, on peut choisir de respecter la durée d'origine de la note, de l'adapter au tempo maître, ou encore de l'ignorer, et jouer la tranche jusqu'à la venue d'un événement (jeu d'un nouvel état, message d'extinction de toutes les notes, etc).

4.2 Mode note à note

Ce mode correspond à une interaction où l'utilisateur a le contrôle sur la machine (par rapport à quand la machine joue). Chaque note ou groupe de notes (accord) entré(e) par l'utilisateur entraîne une harmonisation immé-

diate (un exemple est illustré en Figure 2). Dans le cadre décrit précédemment, ceci est simplement réalisé en demandant un nouvel état dès qu'une nouvelle entrée est détectée, cet état devant être joué le plus rapidement possible. Dans le cas monophonique, cela correspond à chaque nouvelle note jouée. Dans le cas polyphonique, la segmentation de l'entrée est plus complexe à gérer. Deux raisons à cela : l'existence inhérente au jeu humain de légers décalages temporels lorsqu'on joue un accord, et la possibilité d'un jeu legato. Le premier cas risque d'entraîner le jeu d'un état associé à la première note jouée dès que celle-ci est détectée, alors que l'événement pertinent ici est constitué par l'accord dans son ensemble. Dans le second cas, l'inverse peut se produire : un jeu legato peut entraîner un chevauchement temporel passager – or seule la dernière note jouée est pertinente, et elle seule doit être prise en compte lors de la recherche d'un état apparié. Dans les deux cas, il est nécessaire, lorsqu'une nouvelle note est détectée, d'attendre un peu de sorte à lever l'ambiguïté. Ceci introduit inévitablement une certaine latence entre le moment où l'utilisateur joue une note et le moment où la machine joue, mal nécessaire si l'on veut que les demandes d'état correspondent à des événements pertinents par rapport au jeu du musicien. D'où l'existence du mode mono qui permet de réduire cette latence. Dans le cas polyphonique, il s'agit de trouver une valeur suffisamment grande pour écarter les fausses demandes, mais suffisamment faible pour garder la sensation d'une réactivité forte, de 'temps réel'.

Par défaut, ce mode utilise la fonction 'notes tenues', et la vélocité du rendu est entièrement déterminée par celle de l'utilisateur. Ceci donne une véritable impression de contrôle à l'utilisateur, dont le jeu se retrouve augmenté en temps réel. La machine joue suivant au plus près le jeu du musicien, y compris sa dynamique. Le mode 'notes tenues' permet également un enchaînement plus fluide entre tranches, et encourage notamment la présence de liaisons entre notes d'une tranche à l'autre. Ceci permet à moindre frais un rendu polyphonique plus riche que le note à note brut : une même note pourra être tenue par la machine tandis que l'utilisateur joue successivement plusieurs notes.

4.2.1 Gestion de l'échec

Comme on l'a brièvement évoqué section 4.1.1, il est tout à fait possible qu'aucun état candidat ne satisfasse la contrainte d'appariement avec l'entrée utilisateur. On se retrouve alors face à ce que l'on appelle une situation

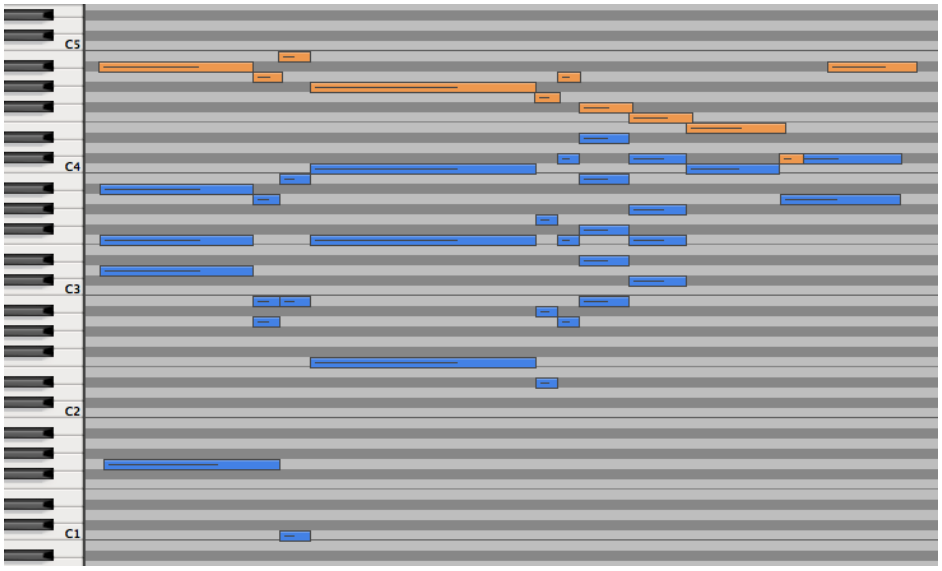

FIGURE 2 – Exemple d'harmonisation de mélodie (mode note à note, avec condition d'appariement *soprano mod12*). L'entrée (en orange) correspond aux premières notes de la pièce pour flûte solo Syrinx de Claude Debussy. Le corpus utilisé est le quatuor à cordes du même compositeur.

d'échec [4]. Plusieurs niveaux d'échecs, et conséquemment d'actions à mener en cas d'échec, sont considérées ici.
(1) Le premier niveau correspond justement au cas où aucun état parmi les sauts possibles ne satisfait la contrainte d'appariement. Dans ce cas, on regarde si la contrainte peut être satisfaite dans les états suivants les états candidats. L'idée sous-jacente, qui reste à évaluer, se base sur l'hypothèse suivante. Appelons i l'état courant, qui contient la note jouée précédemment par l'utilisateur. Si la nouvelle note jouée n'est pas incluse dans l'état $i+1$, l'idée est de regarder dans l'état $i+2$, ce qui suppose implicitement que l'enchaînement $i \rightarrow i+2$ est en moyenne de meilleure qualité qu'un enchaînement $i \rightarrow j$ où j est un état tiré aléatoirement dans le corpus. De même pour tous les sauts possibles.
(2) Si aucune solution n'a été trouvée, on est face à un échec de deuxième ordre. Dans ce cas, plusieurs actions sont possibles. La première possibilité est plutôt une absence d'action, c'est-à-dire que l'on reste où l'on est, et on attend le prochain événement. On peut également choisir de continuer à naviguer dans la mémoire, mais de façon silencieuse. Dans ce cas, un état parmi

4. Malgré les connotations négatives associées à ce terme, un échec n'est pas nécessairement une mauvaise chose d'un point de vue musical, et peut même permettre, dans certaines conditions, d'aérer la réponse musicale de la machine.

les états candidats est sélectionné et devient l'état courant, sans pour autant être joué. Ceci permet de continuer à progresser dans la mémoire musicale, de façon souterraine, de sorte que lorsqu'un état est finalement trouvé, une certaine logique a été conservée. Pour différencier ces deux situations, prenons le cas d'un échec sur une note donnée. Si l'on ne bouge pas, répéter cette note conduira invariablement à une série d'échecs ; dans le cas de la navigation silencieuse, la répétition d'une même note peut à tout moment aboutir à un succès. À noter que l'on peut jouer malgré tout ces états 'souterrains'. Dans le cas où il n'y a que des échecs, ceci n'a évidemment que peu d'intérêt, puisqu'alors la musique générée perd tout lien avec le jeu du musicien. Mais dans le cas où ces échecs sont assez rares, le rendu final peut être tout à fait intéressant, alternant phases de proximité et d'éloignement entre le jeu de la machine et celui du musicien.
(3) Le dernier stade d'échec correspond à une accumulation d'échecs de deuxième ordre. L'idée est qu'au bout d'un certain nombre d'échecs, on peut vouloir agir différemment que ce qu'on a fait jusqu'à présent. On pourrait par exemple tout arrêter, faire une recherche exhaustive, réinitialiser, changer automatiquement certains paramètres, etc. Pour le moment, ce type d'échec est déclenché lorsqu'on a fait face à un échec de deuxième ordre sans avoir eu de succès depuis 500 ms (intervalle de temps réglable par l'utilisateur). On éteint alors toutes les notes jouées par la machine. Ce comportement permet de tolérer, jusqu'à un certain point, une tension momentanée entre le jeu de l'utilisateur et l'accompagnement généré par la machine.

4.3 Génération pure

À l'opposé de l'interaction note à note, pour laquelle la machine n'intervient que lorsque l'utilisateur la sollicite, l'architecture du logiciel permet également à la machine de générer de la musique de façon complètement autonome (de façon similaire à OMax). Pour cela, l'événement qui entraîne la recherche et la lecture d'un nouvel état n'est plus la détection d'une nouvelle note dans le jeu du musicien mais la fin d'une tranche [5]. Les états s'enchaînent alors automatiquement. Si l'état suivant est systématiquement sélectionné, la séquence générée est alors une relecture de la séquence d'origine (avec éventuellement une modification de tempo). Sinon, la machine parcourt la mémoire musicale en empruntant des chemins nouveaux, tout en conservant style et logique de la séquence d'origine, grâce à la présence de motifs communs d'un point à l'autre des sauts.

5. En pratique, tout ceci est anticipé, de sorte à pouvoir profiter du temps de lecture d'une tranche pour effectuer les calculs nécessaires à la recherche d'un nouvel état.

Ce mode est similaire au mode `MIDI_POLY` du logiciel OMax. La différence majeure est la prise en compte du rythme (tempo et pulsation). Grâce à l'association entre l'utilisation d'un tempo maître, la sélection des états en fonction de leur position par rapport à la pulsation (phase) et enfin la possibilité de jouer les états à leur phase d'origine, on obtient une recombinaison du matériel d'origine qui conserve un maintien rythmique cohérent, ce qu'on ne pouvait avoir avec la version actuelle d'OMax. À noter que pour cela il faut évidemment posséder les informations adéquates, à savoir tempo et position par rapport à la pulsation.
Dans le cadre de notre projet, ce mode est intéressant pour tester notamment la qualité de la fonction de classification utilisée (voir section 3.1), indépendamment de la question de l'appariement avec l'entrée musicale.

4.4 Un entre-deux à explorer

Les deux modes de fonctionnement décrits précédemment correspondent à deux extrêmes de l'utilisation du logiciel. Dans le cas du mode note à note, l'événement entraînant la lecture d'une tranche musicale correspond toujours à l'entrée utilisateur. Cette tranche doit par ailleurs satisfaire la condition stricte d'appariement entre son contenu musical et le jeu du musicien. Inversement, dans le cas de la génération pure, l'événement entraînant la recherche d'une tranche musicale est interne à la machine, et cette recherche ne dépend nullement de l'activité de l'utilisateur. Entre ces deux extrêmes, on peut vouloir donner plus ou moins d'autonomie à la machine, à la fois au niveau de quand elle joue (action sur l'événement déclencheur) et au niveau de la contrainte. La version actuelle propose par exemple le méta-mode intermédiaire appelé génération sous-contrainte ('input-constrained generation'). Dans ce mode, la machine génère seulement lorsque l'utilisateur joue, en respectant de façon stricte la contrainte d'appariement. Ainsi, contrairement au mode note à note, lorsque l'utilisateur tient une note, la machine génère un accompagnement en accord avec cette note. Dans ce cadre-là, de nouvelles problématiques s'imposent, en particulier la question de la synchronisation entre le jeu de la machine et celui du musicien, problème important que l'on compte aborder dans les prochains mois.
À noter que d'autres modes d'interaction donnant plus ou moins d'autonomie à la machine sont dès à présent utilisables grâce aux différentes combinaisons possibles entre contrainte stricte (filtre), contraintes plus souples (évaluation) et choix de l'événement déclencheur de la navigation. Diverses expérimentations sont ainsi accessibles à l'utilisateur expérimenté grâce à la partie 'advanced configuration' du logiciel.

5 Tour d'horizon du logiciel actuel : SOMax v0.1

Cette section propose de présenter rapidement l'utilisation du logiciel à travers un parcours rapide de son interface actuelle (visible en Figure 3).

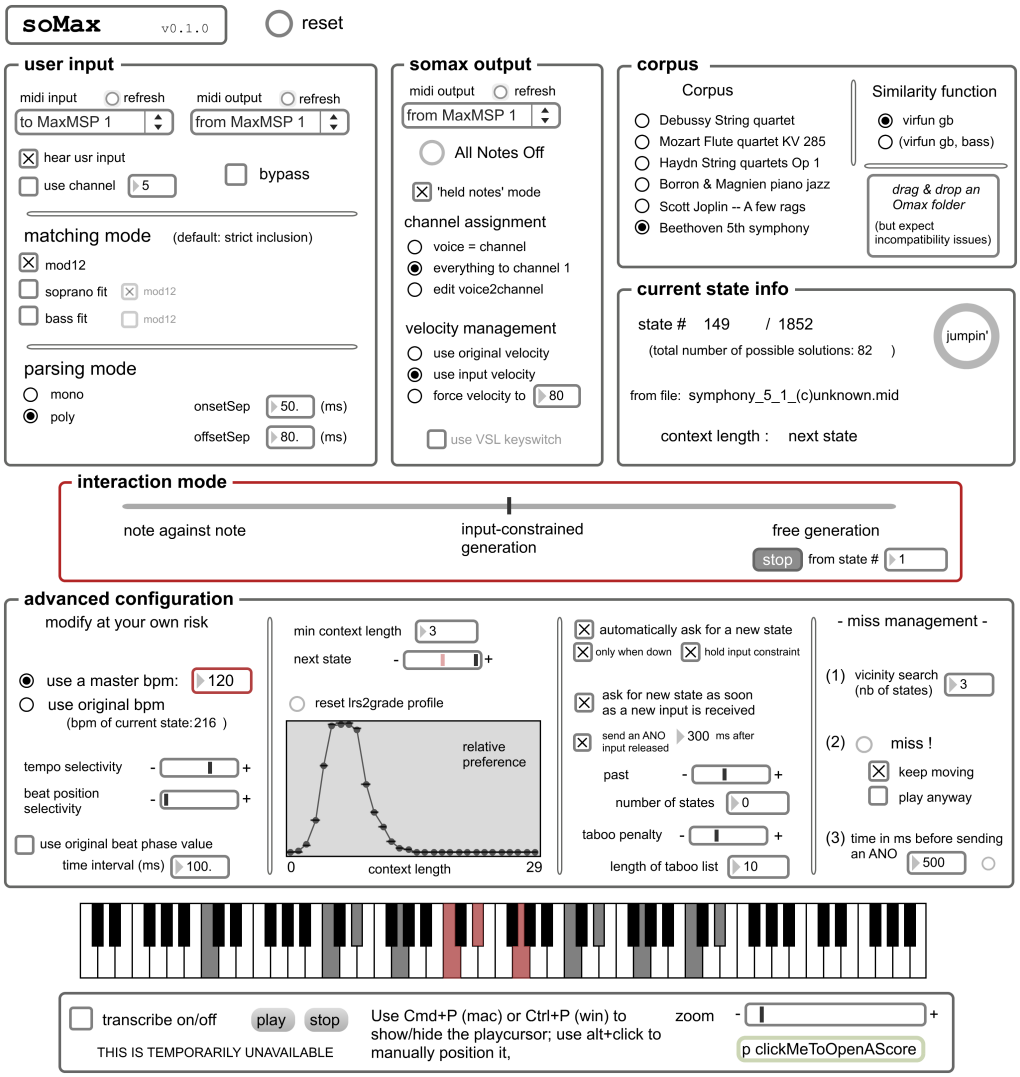

FIGURE 3 – Interface du logiciel SOMax v0.1

5.1 Entrée utilisateur

Cette section permet de configurer le port qui reçoit les informations MIDI de l'utilisateur (provenant par exemple d'un clavier), et le port où ces informations sont envoyées (vers un échantillonneur par exemple).
C'est dans cette section que l'on choisit le mode d'appariement (inclusion stricte, modulo 12, avec contrainte soprano ou basse). À noter que ces différentes options sont combinables.
Enfin, on peut choisir le mode de segmentation de l'entrée utilisateur, soit monophonique, ce qui permet d'avoir une latence la plus faible possible, soit polyphonique, qui permet une gestion polyphonique du jeu du musicien (voir section 4.2).

5.2 Sortie du rendu du logiciel

On choisit ici le port de sortie du matériel musical généré par le logiciel. Le mode 'notes tenues' permet lors de la lecture de prolonger les notes d'une tranche au-delà de leur durée initiale, ce qui permet un contrôle plus proche de cette lecture par l'utilisateur, et encourage la formation de liaisons de notes d'une tranche à une autre (voir section 4.1.6).
L'utilisateur peut ici gérer sur quels canaux les voix du corpus sont envoyées. Par exemple, dans le cas d'un quatuor à corde, les voix sont numérotées de 1 à 4. On peut choisir d'envoyer ces voix sur les canaux de même nom, de tout envoyer sur un même canal (opérant alors une réduction), ou encore de choisir à la main quelle voix sur quel canal.
On peut enfin déterminer de quelle façon la vélocité du rendu est gérée (voir section 4.1.6.

5.3 Corpus

En attente des corpus délivrés par le CNSMD, la version actuelle du logiciel propose plusieurs corpus construits à partir de divers fichiers MIDI trouvés sur Internet [6] ou issus d'un enregistrement des pianistes jazz Magnien et Borron effectué à l'Ircam il y a quelques années.
Ces corpus ont été choisis de manière tout de même à bénéficier de styles musicaux, de tailles, et de complexité différents. Pour le moment, les expériences

6. Disponibles aux adresses : `http://www.kunstderfuge.com/` pour les fichiers MIDI classiques et `http://www.trachtman.org/ragtime/` pour les fichiers MIDI de Scott Joplin. À noter que ces fichiers ne sont pas libres. Ils sont ici utilisés dans un cadre académique confidentiel.

ont été menées sur les corpus suivants :

- Debussy String Quartet
- Mozart flute quartet KV 285
- Haydn String quartets Op. 1
- Borron & Magnien piano jazz
- Scott Joplin – A few rags
- Beethoven 5th symphony

Cette région de l'interface permet également de choisir la fonction de similarité utilisée lors de la construction de l'oracle. Pour le moment, deux fonctions de classification sont à l'étude : virfun gb, et le couple (virfun gb, basse), décrites en section 3.1.

Enfin, une zone permet de déposer un dossier contenant un couple de fichiers aux formats dot et json issus de l'environnement OMax, de sorte à pouvoir récupérer des bases créées avec OMax.

5.4 Information sur l'état courant

Comme son nom l'indique, cette boîte rassemble plusieurs informations sur l'état courant et la façon dont on y est arrivé (numéro de l'état, longueur du contexte, nom du fichier MIDI d'origine de l'état, etc).

5.5 Mode d'interaction

Cette partie, cerclée de rouge, permet de configurer automatiquement différents paramètres du logiciel. Ceci permet à l'utilisateur d'explorer de façon rapide et intuitive différentes possibilités d'interactions proposées par le logiciel. Trois méta-modes sont pour le moment proposés : (1) le mode note à note (voir section 4.2) (2) le mode génération sous contrainte (voir section 4.4) (3) le mode génération pure (voir section 4.3).

5.6 Configuration avancée

S'il est intéressant de pouvoir bénéficier d'une configuration automatique des nombreux paramètres du système, on veut également pouvoir, directement depuis l'interface, avoir un contrôle sur ces paramètres, en particulier dans le cadre exploratoire et expérimental du présent projet.
Sans rentrer dans les détails, ce module permet notamment d'indiquer le tempo global, de choisir la longueur du contexte minimal utilisé lors de la navigation, d'ajuster les poids utilisés lors de l'évaluation des solutions, de

régler le comportement lors d'une situation d'échec, le type d'événement déclencheur de la recherche d'une nouvelle tranche, etc.

5.7 Transcription et visualisation

La partie située tout en bas de l'interface permet de visualiser simultanément sur un clavier le jeu de l'utilisateur (en rouge) et l'augmentation musicale de la machine (en gris). Il est également possible d'ouvrir une partition et de transcrire en temps réel le jeu de l'utilisateur d'une part et le matériel musical généré par la machine d'autre part. Il est nécessaire pour cela d'installer les objets `bach` développés par Andrea Agostini et Daniele Ghisi [7].

6 Résultats & Perspectives

6.1 Succès et limites du prototype actuel

Globalement, le prototype actuel présente d'ores et déjà un intérêt musical certain, et la richesse des comportements possibles est prometteuse. En fonction du corpus et des paramètres (mode de jeu, d'appariement) utilisés, le logiciel peut aller d'un comportement d'harmonisation tonale relativement attendue à un jeu plus surprenant, mêlant autonomie et écoute à la volée. Une interaction inédite entre un musicien et la machine peut dès à présent être expérimentée. Une utilisation prochaine en situation de concert est d'ailleurs envisagée dans le cadre de l'atelier *ImproTech Paris-New York 2012*.

Au-delà de ces premiers succès, de nombreux points sont encore en friche, voire totalement absents du projet. Aucune notion de conduite de voix n'est utilisée actuellement. La logique des sauts étant basée sur une représentation compressée des tranches polyphoniques, ignorant leur disposition verticale, rien ne garantit une quelconque cohérence dans la continuité des voix d'une tranche à l'autre. S'il est relativement facile et direct d'ajouter des règles du type 'éviter les quintes parallèles' pour privilégier tel saut par rapport à un autre selon qu'il respecte ces règles ou non, il sera plus intéressant d'étudier la possibilité d'extraire automatiquement les préférences de conduite de voix présentes dans le corpus, de sorte à conserver au maximum le caractère agnostique du logiciel – qui constitue à mon sens l'un des atouts du projet OMax. Si la prise en compte des données de pulsation et de tempo permet déjà des recombinaisons rythmiquement cohérentes, une prise en charge du rythme

7. Disponibles à l'adresse : `http://www.bachproject.net/`

plus complète est encore à explorer. Par exemple, aucune notion de métrique n'est pour le moment prise en compte, ce qu'une écoute rapide confirme de façon évidente. De plus, lorsque la machine génère de la musique, aucune synchronisation rythmique avec le jeu du musicien n'est à l'œuvre, ce qui limite inévitablement l'interaction.

6.2 À venir

La version actuelle du logiciel se base sur les objets OMax 4.5 (Lévy, 2009), qui – n'étant pas prévus pour un tel usage – ne permettent pas notamment de gérer le rythme. Une utilisation détournée de ces objets permet finalement d'enregistrer et d'utiliser une partie des informations rythmiques nécessaires au présent projet, mais ce bricolage présente de nombreuses limites. Au-delà du manque de propreté à utiliser ces objets bricolés, certaines informations ne peuvent être utilisées, d'où l'absence actuelle de prise en compte de la métrique. Le développement de nouveaux objets plus adaptés au présent projet, développement effectué en collaboration avec Benjamin Lévy, va permettre une meilleure prise en charge et une meilleure utilisation des informations rythmiques, ce qui permettra en particulier d'aborder la question de la métrique. Ces nouveaux objets permettront également, dans le style d'OMax, de créer des bases directement dans le logiciel, à la volée, ou encore d'enrichir des bases existantes.

Comme on l'a vu, au-delà du mode note à note et du mode de génération pure, le logiciel permet de générer de la musique tout en tenant compte, en temps réel, du jeu du musicien (voir section 4.4). Cette possibilité, on l'a dit, est très prometteuse, et sera explorée de façon plus approfondie à l'avenir. Un des points essentiels pour permettre une interaction intéressante est de pouvoir synchroniser l'horloge interne de la machine avec le jeu du musicien. Il faut pour cela être capable d'extraire du jeu du musicien en temps réel les informations liées à la pulsation, ce qui constitue en général un problème difficile. On pourra se baser sur un travail effectué auparavant (Bonnasse-Gahot, 2010). À noter qu'un outil de détection et de suivi du tempo permettrait également d'annoter automatiquement un corpus dépourvu d'informations rythmiques (issu par exemple d'un enregistrement live).

Le prototype actuel va permettre aux partenaires du CNSMDP d'expérimenter les différentes possibilités (corpus, type de fonction de classification, modes de jeu, modes d'appariement, etc.) offertes par le logiciel. Ceci va rendre possible l'évaluation de la pertinence des nombreux paramètres impli-

qués dans le comportement du logiciel ainsi que l'identification de nouvelles limitations. De plus, couplé à l'analyse des classes d'équivalence présentée en section 3.1, l'utilisation du prototype va permettre l'étude, la conception et l'évaluation de nouvelles fonctions de classification (voir section 3.1).
Concernant les corpus utilisés, la souscription récente au site `http://www.kunstderfuge.com/` va permettre de proposer rapidement plus de choix, et de travailler sur des corpus de plus grande taille. Ce dernier point est très important dans le cadre de notre projet. Sachant qu'il s'agit de naviguer sous contraintes, plus la mémoire est grande plus on a de chances a priori de satisfaire ces contraintes, tout en gardant un certain critère de qualité dans le choix des sauts possibles. À noter toutefois que les premiers tests effectués sur des corpus de plus de 100000 états présentent un déclin de performance (la machine 'rame'). Ceci devra être examiné plus en détails.

Signalons enfin que la reflexion autour de l'idée d'accorder la logique du jeu du musicien et celle de la mémoire devra être approfondie (ceci est évoqué à travers la notion de passés communs, voir section 4.1.3), en particulier dans le cas où la machine déroule sa propre temporalité. De même, et la question est liée, la question d'appariement entre jeu du musicien et contenu musical de la mémoire, aujourd'hui simplement traitée en terme d'inclusion, devra être explorée.

Références

Allauzen, C., Crochemore, M. et Raffinot, M. (1999). Factor oracle : a new structure for pattern matching. *In* Pavelka, J., Tel, G. et Bartosek, M., éditeurs : *Proceedings of SOFSEM'99, Theory and Practice of Informatics*, Lecture Notes in Computer Science 1725, pages 291–306, Milovy, Czech Republic. Springer-Verlag, Berlin.

Assayag, G. et Bloch, G. (2007). Navigating the oracle : a heuristic approach. *International Computer Music Conference '07 (Copenhagen, Denmark)*, pages 405–412.

Assayag, G., Bloch, G. et Chemillier, M. (2006). Omax-ofon. *Sound and Music Computing (SMC)*.

Assayag, G. et Dubnov, S. (2004). Using factor oracles for machine improvisation. *Soft Comput.*, 8(9):604–10.

Assayag, G., Dubnov, S. et Delerue, O. (1999). Guessing the composer's mind : Applying universal prediction to musical style. *In Proceedings of the ICMC : International Computer Music Conference, Beijing*, pages 496–499.

Bonnasse-Gahot, L. (2010). Donner à omax le sens du rythme : vers une improvisation plus riche avec la machine. Rapport technique, EHESS-Ircam.

Dubnov, S. et Assayag, G. (1998). Universal classification applied to musical sequences. *In Proceedings of the ICMC : International Computer Music Conference, Ann Arbor, Michigan*.

Dubnov, S., Assayag, G., Lartillot, O. et Bejerano, G. (2003). Using machine-learning methods for musical style modeling. *IEEE Computer*, 10(38): 73–80.

Lefebvre, A., Lecroq, T. et Alexandre, J. (2002). Drastic improvements over repeats found with a factor oracle. *In* Billington, E., Donovan, D. et Khodkar, A., éditeurs : *Proceedings of the 13th Australasian Workshop on Combinatorial Algorithms*, pages 253–265.

Lévy, B. (2009). Visualising omax. Mémoire de D.E.A., ATIAM, UPMC-IRCAM.

Poirson, E. (2002). Simulation d'improvisations à l'aide d'un automate de facteurs et validation expérimentale. Mémoire de D.E.A., ATIAM, UPMC-IRCAM.