Une approche interactive pour la transcription rythmique dans OpenMusic

Adrien Ycart, Jean Bresson, Florent Jacquemard, Slawek Staworko

UNE APPROCHE INTERACTIVE POUR LA TRANSCRIPTION RYTHMIQUE DANS OPENMUSIC

Adrien Ycart[1], Jean Bresson[1], Florent Jacquemard[1,2], Sławek Staworko[3]
[1]Sorbonne Universités, UMR STMS (Ircam, CNRS, UPMC), Paris, France.
[2]Inria Paris, France. [3]University of Lille 3, France and University of Edinburgh, Scotland.
{ycart,bresson}@ircam.fr, {florent.jacquemard, slawomir.staworko}@inria.fr

RÉSUMÉ

Nous présentons un système pour la transcription rythmique intégré dans l'environnement de composition assistée par ordinateur OpenMusic. La transcription rythmique consiste à traduire une suite d'événements datés vers une représentation pulsée et structurée liée à la notation musicale traditionnelle. Notre système privilégie les interactions avec l'utilisateur afin de trouver un équilibre satisfaisant entre différents critères, en particulier la précision de la transcription et la lisibilité des partitions produites. Ce système s'appuie sur une approche uniforme, utilisant des représentations hiérarchiques de la notation de durées sous forme d'arbres de rythmes, et des algorithmes efficaces pour l'énumération paresseuse de solutions de transcription. Sa mise en œuvre se fait via une interface dédiée permettant l'exploration de l'espace des solutions, leur visualisation et leur édition locale, avec une attention particulière portée sur le traitement des appoggiatures.

1. INTRODUCTION

La transcription rythmique est l'action de transformer un flux temporel, par exemple une séquence de notes, en une partition musicale en notation occidentale traditionnelle. Ce flux de notes peut par exemple provenir du jeu d'un musicien, dont on chercherait à retranscrire la performance, ou bien être généré par un procédé de composition algorithmique, par exemple à l'aide d'un logiciel de composition assistée par ordinateur. Nous nous intéressons plus particulièrement au deuxième cas dans cet article.

Ce défi ancien en informatique musicale [17] se décompose classiquement en plusieurs sous-problèmes tels que l'estimation de tempo ou l'estimation de métrique, dont certains sont devenus des problèmes de MIR à part entière [3, 12]. En notation musicale traditionnelle, les durées d'événement sont exprimées en fractions d'une unité dite pulsation ou *temps*, définie par le tempo. La transcription nécessite donc une conversion de durées exprimées en temps physique (par exemple en secondes) vers des durées en unité de temps musical, ce qui passe par l'*inférence d'un tempo* (en temps par minute). De plus, les durées admissibles en notation traditionnelle sont en nombre restreint, et définies par des applications successives de divisions entières du temps (croches, double-croches, etc.) La transcription impose donc d'approximer les durées d'entrée (après conversion en temps) vers des durées correspondant à des figures de notes admissibles. Nous appellerons cette dernière tâche *quantification rythmique*. La transcription peut également être facilitée par une étape préalable de *segmentation*, procédant à un découpage du flux de notes d'entrée en petites unités, plus faciles à analyser. En particulier, délimiter dans le flux de notes des régions de tempo constant permet d'obtenir de meilleurs résultats pour l'inférence de tempo.

Une difficulté de la transcription rythmique provient de la nécessité du couplage entre les tâches d'inférence de tempo et de quantification. D'une part, on ne peut pas quantifier les durées sans connaître le tempo. D'autre part, la qualité de l'inférence d'un tempo ne s'évalue qu'après avoir obtenu le résultat de la quantification. On a donc là un problème de la poule et de l'œuf [7].

Au delà de ce paradoxe, une difficulté importante rencontrée avec la tâche de quantification rythmique elle-même est qu'elle ne définit pas un problème univoque : pour une suite de durées en entrée, différentes notations seront admissibles, ainsi que différents critères pour estimer la validité de ces notations. Un premier critère est par exemple la *précision* de l'approximation (dont résulte la notation), c'est à dire la distance entre la sortie et l'entrée. Un second critère important est la *complexité* de la notation obtenue, c'est à dire à quel point elle sera facilement lisible par un musicien (ou par le compositeur lui-même). Ces deux critères sont souvent antagonistes (cf. figure 1) : en général, plus la notation est précise et plus elle sera difficile à lire. Ainsi, pour fournir des résultats satisfaisants, la quantification devra être obtenue par un bon compromis entre les différents critères.

De plus, une même suite de durées peut être exprimée par des figures de notes différentes, comme le montre l'exemple de la figure 2. Différentes notations représentant la même suite de durées peuvent en effet avoir des sens musicaux différents, et être interprétées différemment.

Toutes ces ambiguïtés font de la quantification rythmique un problème complexe, et il est difficilement concevable de développer une solution entièrement automatique permettant de réaliser les compromis nécessaires entre pré-

CE TRAVAIL A ÉTÉ SUPPORTÉ PAR LE PROJET ANR-13-JS02-0004 "EFFICACE" ET L'UPI "RYTHME" DE L'IRCAM.

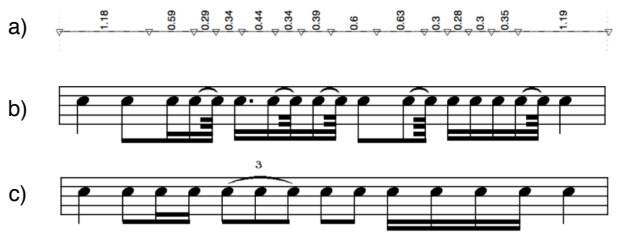

Figure 1. Figure extraite de [9] : a) Séquence d'entrée à quantifier. b) Une notation précise, mais complexe de la séquence. c) Une notation moins précise mais moins complexe de la même séquence.

Figure 2. Deux notations différentes correspondant à la même suite de durées.

cision et complexité, tout en respectant, par exemple (dans le cas de la composition assistée par ordinateur), le message qu'a voulu transmettre le compositeur. De plus, une approche retournant une seule solution prétendument *optimale* risque d'être peu satisfaisante dans de nombreux cas. En effet, il n'existe pas de compromis idéal ou universel entre les critères, et dans un scénario où l'utilisateur se voit imposer une solutions unique, celui-ci devra alors, pour modifier celle-ci, jouer avec les paramètres qui lui sont proposés, sans toujours avoir la visibilité nécessaire.

Dans cet article, nous présentons une approche intégrée dans l'environnement de composition assistée par ordinateur OpenMusic [4], consistant à traiter la transcription comme un problème d'*énumération* multi-critères. Notre objectif est d'appliquer des algorithmes pour énumérer différentes notations possibles, de la meilleure à la pire, selon certains critères donnés. Cette approche diffère d'un traitement de la transcription comme un problème à solution unique. À l'opposé, nous étudions des approches supervisées, interactives, où l'utilisateur guide l'algorithme tout au long du processus, pour finalement converger vers une solution.

Une première étape est la construction d'une structure guidant l'énumération, suivant un *schéma* donné par l'utilisateur. Intuitivement, le schéma décrit la manière dont peuvent être découpés les temps, c'est à dire quelles sont les durées admissibles et comment elles peuvent s'enchaîner. Il s'agit donc d'un langage formel. Ensuite, nous appliquons un algorithme énumérant toutes les solutions produites par les différents découpages définis par le schéma, rangé par un ordre défini par des critères de qualité.

Après une revue de précédents travaux dans le domaine, nous définissons en section 3 les schémas utilisés, en justifiant ce choix. En section 4, nous décrivons les critères de qualité et l'algorithme d'énumération, ainsi que les structure de données utilisées. Le nombre de solutions pouvant être très grand, nous ne pouvons procéder naïvement par un calcul de toutes les solutions, puis leur ordonnancement. L'algorithme applique une méthode paresseuse permettant de fournir les meilleures solutions par paquets de taille donnée (approche dite *k-best*). Nous décrivons également comment nous adaptons cette méthode pour permettre un couplage entre la détection de tempo et la quantification, ainsi que le traitement particulier des appogiatures. Dans la section 5, nous présentons les scénarios de transcription que rend possible notre outil, et son interface utilisateur.

2. ÉTAT DE L'ART

De nombreux quantificateurs existent sur le marché, intégrés dans les éditeurs de partitions ou séquenceurs musicaux (typiquement, pour représenter des données MIDI – mesurées en millisecondes – sous forme de partition). Dans la majorité des cas cependant, les résultats de la transcription rythmique sont peu satisfaisants lorsque les séquences temporelles à traiter sont irrégulières ou trop complexes. L'utilisateur a par ailleurs peu de moyens d'influer sur ces résultats, à moins de les retoucher manuellement a posteriori.

Certains systèmes (comme par exemple l'algorithme décrit par Declan Murphy dans [16]) se basent sur les rapports entre les durées successives, qui doivent être un rapport d'entiers les plus petits possibles. Ali Cemgil et al. ont proposé un modèle Bayésien pour la transcription rythmique [8], dans lequel les auteurs utilisent un modèle de performance où l'erreur entre l'entrée et le rythme idéal est un bruit Gaussien. L'outil actuel de quantification rythmique dans OpenMusic, *omquantify* [15], aligne les instants de début de notes sur des grilles uniformes à l'intérieur de chaque temps, et choisit la grille qui offre le meilleur compromis entre précision et complexité. Ces différents systèmes ont des propriétés intéressantes, mais tous proposent une solution unique : si le résultat n'est pas satisfaisant, il faut relancer l'algorithme avec d'autres paramètres.

La bibliothèque OMKant [15], qui n'est plus présente aujourd'hui dans les versions récentes d'OpenMusic, proposait une approche semi-supervisée pour la segmentation et la transcription rythmique. L'utilisateur pouvait segmenter à la main ou automatiquement le flux d'entrée avec différents algorithmes. Un algorithme de détection de tempo variable permettaient de délimiter les temps, et la quantification des segments se faisait à l'aide d'*omquantify*. Une interface permettait en particulier de spécifier et visualiser différents paramètres (comme par exemple les marqueurs utilisés pour la segmentation), et de choisir entre différentes valeurs de tempo proposées par l'algorithme.

Un nouveau cadre pour la segmentation et l'analyse partitions dans OpenMusic a été proposé dans [6], et peut également être utilisé pour la transcription rythmique. Il permet en particulier de segmenter une séquence et d'utiliser le quantificateur *omquantify* avec des jeux de paramètres différents sur chaque segment.

3. QUANTIFICATION PAR GRILLES NON-UNIFORMES

3.1. Définition du problème de la quantification

Nous considérons des flux de notes monophoniques, c'est à dire que deux notes ne se chevauchent jamais. Dans ces conditions, le flux de notes peut être ramené à une suite d'instants, correspondant chacun soit à un début de note, soit à un début de silence, définis dans un intervalle ouvert $p = [x, x'[$. Nous prendrons donc en entrée une suite croissante d'instants $(x_1, \ldots, x_n)$ dans l'intervalle p, et nous cherchons à obtenir en sortie une autre suite d'instants $(y_1, \ldots, y_n)$ dans le même intervalle, telle que chaque y_i appartient à une suite $(z_0, \ldots, z_m)$ de points de p, avec $m \geq 1$, que nous nommerons *grille*, et telle que $z_0 = x$ et $z_m = x'$. Nous appellerons *segment* l'intervalle $[z_i, z_{i+1}[$ ouvert à droite entre deux points de la grille. Une grille est dite *triviale* si $m = 1$ et *uniforme* si tous ses segments ont la même longueur, c'est à dire si $z_1 - z_0 = z_2 - z_1 = \ldots = z_m - z_{m-1}$.

3.2. Grilles uniformes

Quelles que soient les données que l'on cherche à quantifier, l'une des méthodes de quantification les plus simples consiste à aligner chaque valeur d'entrée x_i sur le point le plus proche dans une grille uniforme. L'algorithme de quantification correspondant consiste à tester toutes les grilles uniformes pour sélectionner celle donnant le résultats le plus satisfaisant. Comme le nombre de grille uniforme est fixe et petit (typiquement 32), cet algorithme a une complexité linéaire. Mais il est incomplet et pourra donner des résultats inutilement compliqués dans le cas où la densité de points en entrée n'est pas uniforme – cf. figure 3.

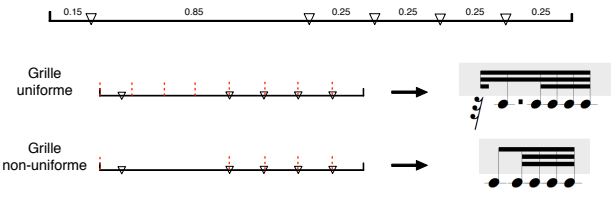

Figure 3. Quantification avec une grille uniforme (division par 8) et une grille non-uniforme (division par 2 puis re-division par 4 de la seconde moitié seulement). La grille non-uniforme donne un meilleur résultat car le pas de la grille est adapté à la densité de points en entrée.

3.3. Schémas de subdivision

Pour plus de complétude, nous utilisons des grilles non-uniformes, définies par subdivisions récursives de segments en parties égales, suivant ce que l'on appelle un *schéma de subdivision*. Un schéma de subdivision indique, pour chaque segment, les divisions uniformes ou suites de divisions uniformes qui sont autorisées. Il est défini comme un graphe orienté acyclique (DAG) ayant une seule racine, et dont chaque arête est étiquetée par un entier. Une grille est valide pour un schéma si et seulement si elle est obtenue par divisions successives de ses segments définies en parcourant les arêtes du DAG depuis la racine. Plus précisément, une grille g est valide pour le schéma $\mathcal{S}$ si (i) d est triviale ou bien si (ii) il existe une arête e étiquetée par ℓ sortant de la racine de $\mathcal{S}$, g est la concaténation de ℓ grilles $g_1, \ldots, g_\ell$ de même longueur et telles que chaque g_i, $1 \leq i \leq \ell$, est valide pour le sous-graphe de $\mathcal{S}$ ayant pour racine le nœud cible de e. Des exemples de schémas et grilles valides sont donnés à la figure 4.

Intuitivement, à chaque étape de la définition d'une grille valide ci-dessus, on peut choisir d'arrêter les divisions (cas i) ou bien de diviser le segment courant en descendant dans le schéma (cas ii). On aura donc un nombre exponentiel de grilles non-uniformes valides pour un schéma donné. Cela permet d'obtenir une grande variété de résultats de transcription différents définis comme les alignements $(y_1, \ldots, y_n)$ de l'entrée $(x_1, \ldots, x_n)$ sur toutes les grilles valides. Bien sûr, le problème réside alors dans l'exploration de l'ensemble de ces résultats ; notre objectif est de sélectionner les meilleurs résultats (selon certains critères) sans être obligés de construire tous les résultats possibles. Nous décrivons dans les sections suivantes les outils utilisés dans ce but.

3.4. Grammaire hors-contexte d'un schéma

La définition récursive des grilles valides pour un schéma $\mathcal{S}$ suggère une représentation arborescente de ces grilles non-uniformes (la division de la grille g en sous-grilles $g_1, \ldots, g_\ell$ correspondant à un branchement de degré ℓ dans l'arbre). Suivant l'approche de Lee [14] (voir aussi la figure 5), on utilise comme représentations les arbres de dérivation d'une grammaire hors-contexte $\mathcal{G}_\mathcal{S}$, définie à partie de $\mathcal{S}$, et qui génère les suites de durées admissibles suivant $\mathcal{S}$ (*i.e.* suites de segments composant une grille valide pour $\mathcal{S}$).

Les symboles non-terminaux de $\mathcal{G}_\mathcal{S}$ sont de la forme $[a_d^*]$. Ils sont définis de sorte que $(*)$ à un intervalle $[x, x'[$ et un non-terminal, on peut associer de manière unique un sous-intervalle dans $[x, x'[$. Par exemple, pour $[0, 1[$ et $[3_2]$, le sous-intervalle est le deuxième tiers de l'intervalle de départ : $[\frac{1}{3}, \frac{2}{3}[$. Pour un non-terminal de longueur supérieure à 1, on poursuit récursivement : pour $[3_2 2_1]$, il s'agit de la première moitié du précédent intervalle, $[\frac{1}{3}, \frac{1}{2}[$. La suite des sous-intervalles correspond aux segments d'une grille valide.

La grammaire de la figure 6 est associée au schéma de subdivision de la figure 4(A). On notera que l'on aurait pu construire plus efficacement une grammaire caractérisant les grilles valides pour un schéma donné. Cependant, la propriété $(*)$ est indispensable dans notre cas (cf. section 4).

Par souci d'efficacité et pour éviter les solutions redondantes, certains non-terminaux sont omis dans $\mathcal{G}_\mathcal{S}$: nous interdisons de redécouper un segment ne comportant aucun x_i puisque dans ce cas, raffiner la grille ne changera rien à l'alignement.

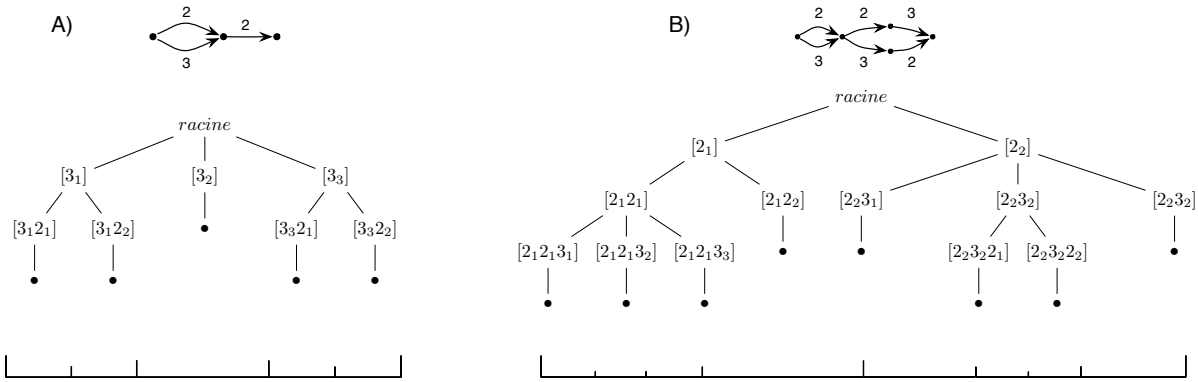

Figure 4. Deux exemples de schémas de subdivision. Pour chaque schéma, un exemple d'arbre de dérivation est présenté, ainsi que la grille non-uniforme qui lui est associée.

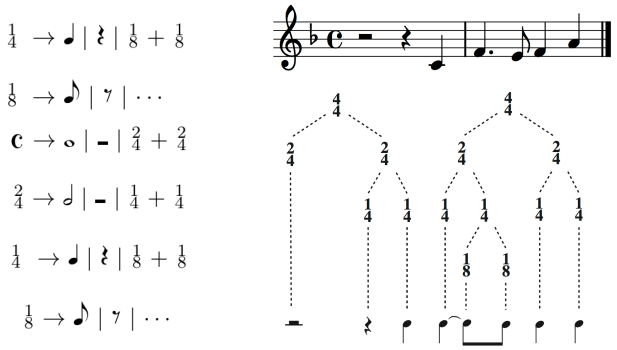

Figure 5. Une grammaire hors-contexte, un rythme et l'arbre de dérivation correspondant [14].

$$
\begin{array}{rcl}
racine & \rightarrow & [2_1]\,[2_2] \mid [3_1]\,[3_2]\,[3_3] \mid \bullet \\
[2_1] & \rightarrow & [2_12_1]\,[2_12_2] \mid \bullet \\
[2_2] & \rightarrow & [2_22_1]\,[2_22_2] \mid \bullet \\
[3_1] & \rightarrow & [3_12_1]\,[3_12_2] \mid \bullet \\
[3_2] & \rightarrow & [3_22_1]\,[3_22_2] \mid \bullet \\
[3_3] & \rightarrow & [3_32_1]\,[3_32_2] \mid \bullet
\end{array}
$$

$$
\begin{array}{rcl@{\qquad}rcl}
[2_12_1] & \rightarrow & \bullet & [3_12_1] & \rightarrow & \bullet \\
[2_12_2] & \rightarrow & \bullet & [3_12_2] & \rightarrow & \bullet \\
[2_22_1] & \rightarrow & \bullet & [3_22_1] & \rightarrow & \bullet \\
[2_22_2] & \rightarrow & \bullet & [3_22_2] & \rightarrow & \bullet \\
 & & & [3_32_1] & \rightarrow & \bullet \\
 & & & [3_32_2] & \rightarrow & \bullet
\end{array}
$$

Figure 6. Grammaire hors-contexte associée au schéma de subdivision de la figure 4-A).

3.5. Arbres de rythme et séries d'arbres

Les représentations arborescentes de rythmes sont établies depuis longtemps, en particulier dans OpenMusic [2] (voir figure 7). Elles correspondent en effet naturellement à la définition, en notation traditionnelle, du rythme par des divisions successives d'une unité de temps.

Chaque résultat de transcription $(y_1, \ldots, y_n)$ sera converti en un arbre de rythme afin d'être affiché dans OpenMusic. On rappelle que $(y_1, \ldots, y_n)$ est défini comme l'alignement au plus proche de l'entrée $(x_1, \ldots, x_n)$ sur une grille valide pour le schéma $\mathcal{S}$. La grille valide étant un arbre de dérivation de $\mathcal{G}_\mathcal{S}$, la construction de l'arbre de rythme (que nous ne détaillerons pas ici) se fait par transformation d'arbres.

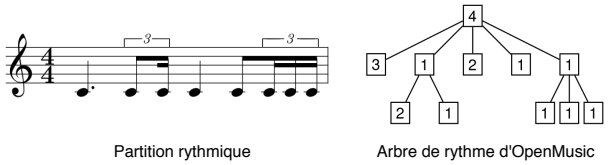

Figure 7. Un rythme et son équivalent sous forme d'arbre de rythme dans OpenMusic. Dans ce formalisme, l'étiquette de chaque nœud représente sa durée relative par rapport à ses frères, et le nombre de fils de chaque nœuds indique en combien de parties la durée du père est divisée.

4. ÉNUMÉRATION MULTI-CRITÈRES

Afin de classer les résultats de quantification, nous considérons la notion de *série d'arbres* [10], qui est une fonction associant à tout arbre (de rythme) une valeur de poids dans un domaine donné – ici les réels. Pour l'énumération d'une telle série, nous utilisons un algorithme récursif, avec des techniques de programmation dynamique. L'évaluation paresseuse des poids permet de bonnes performances malgré la taille exponentielle de l'espace de recherche.

Nous détaillons ci-dessous le calcul de la fonction de poids, qui prend en compte différents critères, puis nous décrirons l'algorithme d'énumération.

4.1. Fonction de poids

4.1.1. Critères de qualité et combinaison

La fonction de poids associe une valeur réelle à une suite d'instants à quantifier $(x_1, \ldots, x_n)$ et une grille valide (c'est à dire un arbre de dérivation). Plus cette valeur

est faible, plus l'arbre de rythme correspondant est considéré comme une bonne transcription de la suite d'instants à quantifier.

Le poids d'un arbre est calculé en combinant différents critères, qui sont des fonctions qui associent à une suite d'instants et une grille valide une valeur réelle. Nous prenons en compte des critères de distance d'une part, de complexité d'autre part. Les critères sont calculés récursivement : la valeur d'un critère pour un arbre est évaluée à partir des critères de ses fils.

Comme critère de *distance*, nous avons choisi la somme des écarts (en valeur absolue) entre les instants de début et de fin des notes en l'entrée et en sortie de l'algorithme, c'est à dire la distance entre un instant d'entrée et le point de la grille sur lequel il est aligné.

Pour un sous-arbre t qui est une feuille, cette valeur n'est calculée que pour les instants d'entrée qui sont contenus dans l'intervalle $segment(t)$ correspondant à t (cf. section 3.4).

$$dist(t) = \sum_{x_i \in segment(t)} |x_i - y_i| \qquad (1)$$

Intuitivement, si t est une feuille, cela veut dire que l'on ne subdivise pas plus finement l'intervalle correspondant $segment(t)$, et les points contenus dans cet intervalle sont alors directement alignés sur la borne la plus proche.

Pour un arbre $t = a(t_1, \ldots, t_\ell)$, $dist$ est obtenu en sommant les $dist$ des fils (les segments des fils forment une partition du segment du père).

$$dist\big(a(t_1, \ldots, t_\ell)\big) = \sum_{i=1}^{\ell} dist(t_i) \qquad (2)$$

Le critère de *complexité* retenu est une combinaison de différents sous-critères. Le premier est relatif à la taille de l'arbre et aux arités des différents nœuds. Il s'agit de la somme des nombres de nœuds ayant une certaine arité, pondérée des coefficients. Nous notons β_j le coefficient décrivant la complexité de l'arité j, et suivons l'ordre préconisé dans [1] pour classer les arités de la plus simple à la plus complexe : $\beta_1 < \beta_2 < \beta_4 < \beta_3 < \beta_6 < \beta_8 < \beta_5 < \beta_7$... L'autre sous-critère que nous avons choisi correspond au nombre d'appoggiatures présentes dans la notation finale. Une appoggiature correspond au cas où plusieurs points d'entrée x_i sont alignés sur le même point de la grille (nous rappelons que nous considérons des entrées monophoniques, deux notes alignées au même point ne correspondent donc pas à un accord). Nous cherchons à limiter au maximum le nombre d'appoggiatures, considérant que, le plus souvent, elles constituent un indicateur de manque de précision de la grille.

Si t est une feuille, le premier sous-critère est nul et la complexité $comp(t)$ est donc le nombre d'appoggiatures, noté $g(t)$, qui est déterminé en comptant le nombre de points du segment alignés sur chaque borne (voir à ce sujet la discussion en section 4.4).

Pour un arbre $t = a(t_1, \ldots, t_\ell)$:

$$comp\big(a(t_1, \ldots, t_\ell)\big) = \beta_\ell + \sum_{i=1}^{\ell} comp(t_i) \qquad (3)$$

Le poids $w(t)$ d'un arbre t est calculé en faisant une combinaison linéaire des critères précédents :

$$w(t) = \alpha.dist(t) + (1 - \alpha).comp(t) \qquad (4)$$

Le coefficient α est un paramètre de l'algorithme, il permet d'ajuster l'importance relative des deux critères dans le résultat final : $\alpha = 0$ donnera des résultats privilégiant la simplicité de la notation (arbre de rythme petit, arités simples, peu d'appoggiatures) au détriment de la fidélité de la transcription, tandis que $\alpha = 1$ donnera des arbres de rythmes de profondeur maximale, souvent moins lisibles, mais retranscrivant le plus précisément possible la suite de durées d'entrée.

4.1.2. Monotonie de la fonction de poids

Les critères ainsi que la fonction permettant de les combiner n'ont pas été choisis au hasard, il ont été choisis pour que le poids vérifie une propriété de monotonie, qui est nécessaire au fonctionnement de l'algorithme d'énumération. Cette propriété est la suivante :

$$\begin{aligned} &\forall t = a(t_1, \ldots, t_\ell)\, \forall i \in [1..\ell]\, \forall t'_i\, w(t'_i) > w(t_i) \Rightarrow \\ &w(a(t_1, \ldots, t_{i-1}, t'_i, t_{i-1}, \ldots, t_\ell) > w(a(t_1, \ldots, t_\ell) \end{aligned} \qquad (5)$$

En d'autres termes, si on remplace un des sous-arbres par un sous-arbre de poids plus grand, le poids du père sera plus grand.

Une condition suffisante pour que cette propriété soit vérifiée est premièrement de définir $dist$ et $comp$ par

$$dist\big(a(t_1, \ldots, t_\ell)\big) = \delta_\ell\big(dist(t_1), \ldots, dist(t_\ell)\big)$$

et

$$comp\big(a(t_1, \ldots, t_\ell)\big) = \eta_\ell\big(comp(t_1), \ldots, comp(t_\ell)\big)$$

où les fonctions δ_ℓ et η_ℓ (calculant les distance et complexité d'un arbre à partir des distance et complexité de ses fils) sont choisies monotones, au sens de l'équation (5), et affines par rapport à chaque sous-arbre. Deuxièmement, on définit la fonction de poids par :

$$w(t) = \rho\big(dist(t), comp(t)\big)$$

où ρ est linéaire. Nous avons donc choisi des fonctions qui se conforment à ces conditions.

4.2. Algorithme d'énumération

On suppose fixé un schéma de subdivision $\mathcal{S}$ et $\mathcal{G}_\mathcal{S}$ sa grammaire associée.

4.2.1. Table dynamique

A partir de la grammaire $\mathcal{G}_\mathcal{S}$, nous construisons une table sur laquelle s'appuiera l'algorithme d'énumération. Les entrées de cette table sont les non-terminaux de la grammaire, et chaque entrée contient deux listes : une liste des meilleurs arbres pour le nœud considéré ainsi que leurs poids (initialisée vide), et une liste de candidats, qui contiendra les arbres parmi lesquels on choisira les meilleurs. Cette seconde liste peut éventuellement contenir des arbres dont les poids n'ont pas encore été évalués.

Les arbres de dérivation ne sont pas stockés in extenso mais sont représentés par des *calculs*, qui sont des listes de pointeurs vers chacun des fils. Le pointeur correspond au rang du sous-arbre dans la liste des meilleurs arbres pour le non-terminal correspondant au fils. Par exemple, pour le non-terminal $[3_1 2_2]$, un calcul (2 1 3) indique que le nœud considéré a 3 fils, celui de gauche est le 2ème meilleur pour le non-terminal $[3_1 2_2 3_1]$, celui du milieu est le meilleur pour le non-terminal $[3_1 2_2 3_2]$, et celui de droite est le 3ème meilleur arbre pour le non-terminal $[3_1 2_2 3_3]$. Un calcul vide indique une transition vers un symbole terminal. En partant de la racine, et en allant jusqu'aux symboles terminaux, on peut ainsi reconstituer l'arbre de dérivation en entier.

4.2.2. Initialisation de la table

L'initialisation et la mise à jour de la liste de candidats tire profit de la propriété de monotonie évoquée à la partie 4.1.2. En effet, cette propriété implique que l'arbre de poids le plus faible sera obtenu en choisissant les fils de poids les plus faibles à chaque nœud. L'arbre de poids le plus faible sera donc systématiquement obtenu avec un calcul du type $(1 \ldots 1)$, et le nombre de 1 dépend de l'arité du nœud.

Pour chaque non-terminal, l'initialisation de la liste de candidats se fait donc en consultant le schéma de subdivision pour savoir quelles arités sont possibles pour ce non-terminal, et en ajoutant à la liste pour chaque arité un calcul du type $(1 \ldots 1)$ de longueur égale à l'arité. Leurs poids sont pour l'instant indéterminés.

4.2.3. Énumération

L'algorithme d'énumération que nous avons développé est basé sur le fonctionnement de ceux décrits par Huang et Chiang [11], et est décrit en détail dans [19]. Il s'agit d'un algorithme récursif, utilisant des techniques de programmation dynamique qui permettent de réutiliser les résultats déjà calculés, et évaluant les poids de manière paresseuse (on n'a pas besoin d'évaluer tous les poids avant de commencer à énumérer les meilleurs).

L'algorithme fonctionne de manière récursive : pour évaluer le poids d'un nœud interne, on va chercher à évaluer le poids de chacun de ses fils.

La fonction principale est la fonction récursive $best$, qui renvoie pour un certain non-terminal son $k^{\text{ème}}$ calcul de poids le plus faible ainsi que son poids. On va donc appeler $best(k, racine)$:

— Si la liste des meilleurs calculs contient déjà k éléments, on retourne directement le $k^{\text{ème}}$ calcul de poids le plus faible.

— Sinon, si pour un non-terminal $[\sigma]$ le poids d'un candidat $(i_1, i_2, \ldots, i_\ell)$ n'est pas évalué, on appelle récursivement $best(i_1, [\sigma\ell_1])$, $best(i_2, [\sigma\ell_2])$, …, $best(i_\ell, [\sigma\ell_\ell])$ pour évaluer chacun de ses fils, puis on calcule son poids.

— Une fois tous les poids évalués, on extrait le calcul de poids le plus faible de la liste des candidats, on l'ajoute à la liste des meilleurs calculs, et on ajoute les n nouveaux candidats suivant, de poids indéterminés : $(i_1+1, i_2, \ldots, i_\ell), (i_1, i_2+1, \ldots, i_\ell), \ldots, (i_1, i_2, \ldots, i_{\ell-1}, i_\ell+1)$

La mise à jour de la liste de candidats utilise la propriété de monotonie. En effet, si le meilleur calcul est $(i_1, i_2, \ldots, i_\ell)$, alors on sait que le deuxième meilleur sera forcément parmi la liste des suivants ci-dessus (à moins qu'une autre subdivision, déjà présente dans la liste des candidats, ne soit plus avantageuse). Ainsi, on peut obtenir les k meilleurs calculs pour la racine, et de là en déduire les k meilleurs arbres de rythme.

4.2.4. Complexité

L'algorithme est efficace à plusieurs titres. D'une part, les poids des arbres déjà calculés sont stockés dans la table, et donc si un calcul $(i_1\ i_2\ \ldots\ i_\ell)$ a été évalué, l'évaluation du calcul $((i_1+1)\ i_2\ \ldots\ i_\ell)$ ne nécessitera d'évaluer que le poids du premier sous-arbre (les autres sont déjà évalués et stockés dans la table). D'autre part, on n'explore pas tout l'espace de recherche : le meilleur calcul est choisi parmi un nombre limité de calculs du type $(1\ \ldots\ 1)$, et à chaque étape, le suivant est choisi en n'évaluant qu'un nombre limité de nouveaux candidats.

On utilise les notations suivantes :

a_{max} : Arité maximale que l'on peut trouver dans le schéma de subdivision

d_{max} : Profondeur maximale du schéma de subdivision (correspond à la longueur du plus long chemin dans le schéma de subdivision)

c_{max} : Nombre maximum de choix d'arité qu'on a pour une subdivision

k : Nombre de solutions que l'on demande à l'algorithme de renvoyer

La complexité de l'obtention des k meilleurs calculs, une fois la table créée, est la suivante :

$$O(\underbrace{(a_{max}c_{max})^{d_{max}+1}}_{\text{évaluation du } 1^{er} \text{ calcul}} + \underbrace{k(a_{max}^{d_{max}+1} log(ka_{max}))}_{(k-1)\text{suivants}}) \quad (6)$$

4.3. Quantification et estimation de tempo

L'algorithme d'énumération que nous avons présenté est appliqué pour fournir k meilleures solutions de quantification, dans un cas où le tempo est supposé connu.

L'estimation du tempo est un problème difficile ; nous proposons une solution (simple) intégrée à notre algorithme de quantification.

L'idée de la phase d'estimation de tempo est, après une étape préliminaire de segmentation, de proposer une ou plusieurs valeurs de tempo qui correspondent à la division de chaque segment en un certain nombre m de parties égales qui correspondront aux pulsations (ce qui suppose un tempo constant sur tout le segment). Les valeurs de m sont choisies de sorte que le tempo soit compris entre 40 et 200 pulsations par minute (ou n'importe quelle plage spécifiée par l'utilisateur) : $\frac{dur \times 40}{60} \leq m \leq \frac{dur \times 200}{60}$, où dur est la durée du segment, en secondes. Pour prendre en compte ces différentes valeurs de pulsation (donc de tempo) dans l'algorithme de recherche, il suffit donc de rajouter un étage au schéma de subdivision $\mathcal{S}$: on ajoute un nœud, qui sera la nouvelle racine, et on relie la nouvelle racine à l'ancienne avec des arrêtes correspondant aux valeurs de m autorisées.

Cette approche a l'avantage de garder le même formalisme, le même algorithme pour l'estimation de tempo et la quantification : on peut donc classer des solutions pour des tempi différents, et donc coupler l'estimation de tempo et la quantification.

Cette méthode offre donc de nombreux avantages, mais fonctionne moins bien sur les segments de trop grande taille. En particulier, on aura beaucoup de tempi possibles (ce qui alourdit considérablement les calculs), et on aura des tempi très rapprochés : par exemple, pour un segment de 2 minutes, $m \in [80; 400]$, ce qui représente 320 valeurs de tempo possibles, toutes écartées de 0,5 pulsations par minutes (une telle résolution n'est pas nécessaire).

Dans notre système, nous utiliserons finalement une méthode légèrement différente : au lieu de n'utiliser qu'un seul schéma avec plusieurs tempi possibles, nous utiliserons un schéma et une table par tempo. En termes de complexité en temps et en espace, les deux approches sont strictement identiques. La différence vient du fait qu'avec la première, l'algorithme classe les solutions tous tempi confondus, ce qui permet d'obtenir la meilleure quantification avec le meilleur tempo. Par contre, on ne peut pas demander explicitement les k meilleures solutions pour un tempo donné. C'est l'inverse avec l'approche que nous avons choisie : chaque tempo est indépendant, il n'y a pas de comparaison entre des quantifications pour des tempi différents, par contre on peut demander les k meilleures solutions pour un tempo particulier.

4.4. Appoggiatures

Une appoggiature correspond au cas où deux points d'entrée sont alignés sur le même point de la grille. Ce cas est traité dans la reconstruction de l'arbre de rythme (une extension au formalisme d'arbre de rythme d'OpenMusic a été implémentée pour les prendre en compte), mais il subsiste des problèmes dans le comptage de ces appoggiatures.

Lorsque les deux instants alignés au même point de la grille sont dans le même segment, on sait que dans ce segment, on ne subdivisera pas à nouveau, et les instants seront bien alignés à cet endroit. En revanche une difficulté survient lorsque les deux points sont dans deux segments différents (de part et d'autre du point de la grille) : le calcul de $comp$ est fait uniquement à partir des fils, sans rien supposer sur les segments adjacents (c'est justement ce qui fait l'efficacité de notre algorithme – les calculs faits sur un segment pourront être réutilisés quelle que soit la configuration dans laquelle il se trouve). Si un instant est aligné à droite du segment dans lequel il se trouve, on ne sait donc pas si dans le segment qui se situe directement à sa droite, un instant sera aligné à gauche ou non : cela dépend du nombre de subdivisions qui ont été faites dans le segment de droite. Un exemple est présenté en figure 8. Lorsque l'on fait simplement la somme des appoggiatures dans chaque segment, on ne compte pas les cas où deux points provenant de deux segments différents sont alignés au même point (ici, dans chacun des segments, $g = 0$ donc à la racine, on a également $g = 0$). Comme certaines appoggiatures ne sont pas comptabilisées par l'algorithme, elles ne sont pas pénalisées, c'est pourquoi il arrive de retrouver, parmi les premières solutions proposées par l'algorithme, des solutions contenant des appoggiatures. S'il souhaite les éliminer, l'utilisateur devra lui-même éditer la solution.

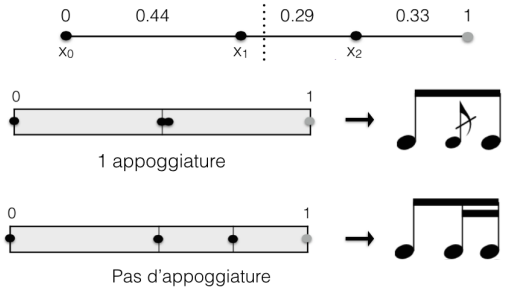

Figure 8. Un exemple d'indétermination du nombre d'appoggiatures : dans le premier cas, x_1 et x_2 sont tous les deux alignés au milieu, dans le deuxième cas, x_2 n'est plus aligné au milieu car la grille est plus fine. Dans le segment de gauche, lors de l'évaluation de $comp$, on ne pourra donc pas savoir si x_1 est une appoggiature ou non.

5. BIBLIOTHÈQUE OPENMUSIC ET INTERFACE UTILISATEUR

L'algorithme présenté à la partie précédente a été implémenté sous forme d'une bibliothèque OpenMusic. Une interface a également été développée pour superviser et exploiter les résultats de cet algorithme. Une capture d'écran de cette interface est présentée sur la figure 9.

L'interface permet de segmenter le flux de notes d'entrée, de visualiser les différentes représentations de la suite de notes (durées en secondes, ainsi que les transcriptions), choisir parmi les différentes solutions et les différents tempi proposés et éditer les solutions choisies.

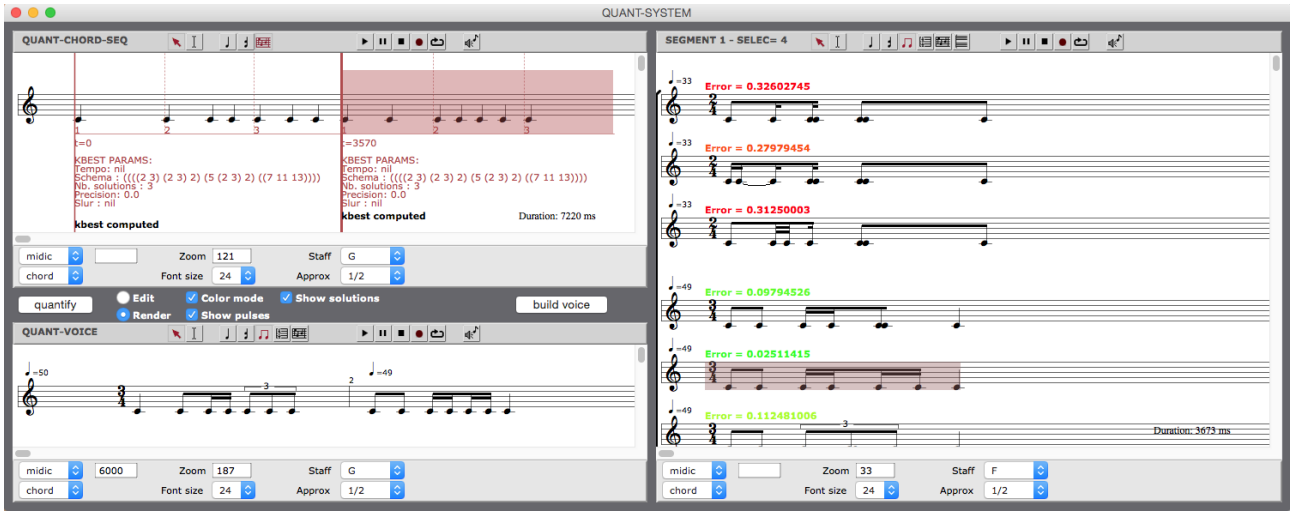

Figure 9. Interface utilisateur développée pour l'algorithme de quantification dans OpenMusic

5.1. Processus de travail

L'utilisateur commence par charger dans le système de transcription un CHORD-SEQ, qui est l'équivalent dans OpenMusic d'un fichier MIDI (les dates de début et de fin des notes sont en millisecondes). La transcription rythmique se fait ensuite en 4 étapes.

Segmentation : L'utilisateur segmente le flux de notes dans le panneau en haut à gauche. Typiquement, la longueur des segments doit être de l'ordre de la mesure. De plus, ces segments doivent de préférence correspondre à des régions où le tempo est constant, pour la raison évoquée à la partie 4.3 : chaque segment sera découpé en m parties égales, qui correspondent aux pulsations. L'utilisateur peut également spécifier différents jeux de paramètres sur chaque segment, comme par exemple le schéma de subdivision, modifier les bornes de tempi admissibles, ou même spécifier un tempo particulier. [1]

Quantification : L'algorithme est lancé sur chaque segment indépendamment, et les k meilleures solutions, pour chaque tempo et pour chaque segment, sont calculées.

Choix d'une solution : L'utilisateur visualise les transcriptions calculées dans le panneau de droite, et sélectionne, pour chaque segment, une de ces transcriptions. Pour aider l'utilisateur à déterminer la meilleure transcription, les valeurs de $dist$ de chaque transcription sont indiquées. Les transcriptions choisies sont ensuite concaténées et affichées dans le panneau en bas à gauche.

Édition de la solution : L'utilisateur peut éditer la solution retenue. Éditer la solution revient à explorer la table utilisée par l'algorithme de quantification. L'utilisateur sélectionne sur la partition un sous-arbre de rythme, et peut visualiser les autres transcriptions contenues dans la liste des meilleurs calculs pour le non-terminal correspondant à la racine de ce sous-arbre. Pour déterminer la qualité de ces transcriptions et trouver le bon compromis entre lisibilité et précision, l'utilisateur peut visualiser $dist$ à l'aide d'un code de couleur. Il peut également demander à l'algorithme de calculer les k solutions suivantes pour ce sous-arbre. Il peut ensuite sélectionner le sous-arbre qu'il souhaite parmi cette liste, et il sera remplacé dans la partition finale.

1 . Un algorithme de segmentation automatique en régions de tempo constant est en cours de développement.

5.2. Affichage de la solution

La solution affichée correspond à l'arbre de rythme obtenu en alignant chaque instant d'entrée au point le plus proche de la grille. Ce faisant, il arrive que des segments de la grille ne contiennent aucun instant d'entrée : ces segments sont donc représentés soit par des silences, soit par des notes liés. Dans le cas de notes liées, cela peut donner lieu à des représentations qui ne sont pas naturelles : par exemple, le rythme "croche pointée-double croche" serait représenté par 4 doubles croches, dont les 3 premières sont liées (cf. figure 10a) La première représentation est évidemment préférable, mais elle a un inconvénient : elle rend impossible l'édition. En effet, la deuxième représentation traduit le fait qu'un segment a été découpé en 4 partie égales. Par exemple si on souhaite raffiner la grille dans le 3ème segment (cf. figure 10b), avec la notation "croche pointée-double croche", on ne pourra pas sélectionner ce segment pour explorer d'autres solutions, puisqu'il a été fusionné avec les deux premiers segments. Au contraire, avec l'autre notation, chaque symbole terminal de l'arbre est représenté par une tête de note, on pourra donc explorer d'autres solutions pour chaque segment.

L'affichage de la solution est donc soumis à deux exigences contradictoires. D'un côté, on souhaite avoir la représentation la plus lisible possible, la plus naturelle possible, et d'un autre côté, on veut conserver la structure

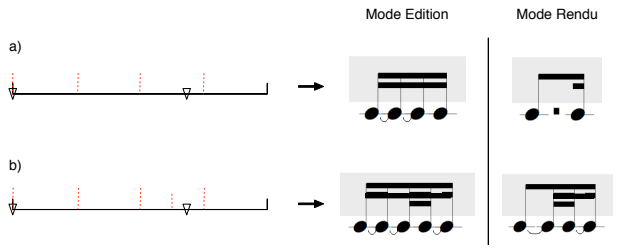

Figure 10. Le même rythme est quantifié avec deux grilles différentes. Le mode Rendu donne une transcription plus lisible, le mode Édition conserve la structure de l'arbre de dérivation.

de l'arbre de dérivation pour pouvoir l'éditer. Pour cette raison, notre interface propose deux modes d'affichage, l'un permettant l'édition, l'autre permettant de visualiser le résultat dans sa forme la plus simple.

6. DISCUSSION

6.1. Evaluation

L'évaluation d'un tel système est une tâche difficile, du fait de l'absence de critères objectifs pour évaluer une "bonne" transcription. Une méthode d'évaluation pourrait être de quantifier des performances de morceaux connus, et vérifier que l'on obtient bien le même résultat que la partition originale. Cette méthode a deux inconvénients. Le premier est que notre approche est interactive : le premier résultat sera rarement le bon, mais on pourrait l'obtenir en quelques opérations. Il faudrait alors évaluer non pas si le résultat est le même que la cible, mais plutôt évaluer le nombre d'opérations nécessaires pour retrouver la cible, ou encore encore évaluer si cette cible se trouve dans les n premières solutions proposées. Le deuxième inconvénient est que notre système n'a pas été conçu pour la quantification de performances, mais plutôt de séries de notes obtenues par un calcul. Pour quantifier des performances, l'hypothèse des segments de tempo constants n'est pas adaptée, il aurait mieux valu inférer un tempo variable. Les résultats ne seront donc pas représentatifs de la qualité de notre système pour son utilisation principale. Des essais sont en cours auprès de compositeurs pour évaluer la pertinence et la facilité d'utilisation du système.

6.2. Choix des paramètres

En règle générale, on obtient des résultats satisfaisants en classant les résultats uniquement selon $comp$ ($\alpha = 0$). Comme les deux critères sont antagonistes, classer selon $comp$ croissant revient approximativement à classer selon $dist$ décroissant (il s'agit d'une observation empirique, pas d'un résultat démontrable). Ainsi, il y a une certaine régularité dans l'ordre de classement des solutions. Lorsque α prend des valeurs autour de 0.5, on peut obtenir des solutions très différentes avec des poids similaires (par exemple des solutions précises et complexes juste à côté de solutions simples et imprécises). L'exploration de l'espace des solutions est moins facile, on a peut-être plus de chances d'obtenir de bonnes transcriptions dans les n premiers résultats, mais on ne sait pas à quoi va ressembler le $n+1^{\text{ème}}$. L'inconvénient de ce classement est que l'on obtient beaucoup de résultats avec des appoggiatures. En effet, on ne peut pas comptabiliser certaines appoggiatures (cf section 4.4), des résultats avec beaucoup d'appoggiatures auront donc un $comp$ très faible, et comme $dist$ ne les pénalise pas, ces transcriptions se retrouveront dans les premières solutions.

6.3. Segmentation

On note également de moins bons résultats lorsque les segments à quantifier sont trop longs, comme évoqué à la section 4.3 (calculs trop longs, beaucoup de tempi très similaires et donc difficulté pour choisir le bon). Plus généralement, le résultat est très sensible à la segmentation, du fait de la méthode d'estimation de tempo utilisée, ce qui pose des limites à l'interactivité de notre système (on ne peut pas visualiser le résultat des modifications faites sur la segmentation sans relancer tout l'algorithme).

7. CONCLUSION ET PERSPECTIVES

Nous avons présenté un nouveau système interactif pour la segmentation et la transcription rythmique. Le système énumère à l'aide d'un algorithme dynamique et paresseux des transcriptions possibles selon un poids prenant en compte la précision et la complexité de la notation. Une interface permet à l'utilisateur de superviser le processus, en choisissant parmi les transcriptions proposées et en les éditant si nécessaire. Cet outil est en cours de test auprès de compositeurs.

La fonction de poids étant le point sensible de notre algorithme, c'est en priorité à ce niveau que différentes améliorations peuvent être envisagées.

Tout d'abord, les coefficients utilisés dans le calcul du poids sont pour l'instant fixés arbitrairement, ou bien laissés comme paramètres présentés à l'utilisateur. En particulier, les β_j dans le calcul de $comp$ sont fixés selon un ordre "logique", mais basé sur aucune vérité scientifique. Pour déterminer des coefficients plus adaptés, nous pourrions nous servir d'une base de couples (performances, partitions) comme par exemple le corpus Kostka-Payne [13] (précédemment utilisé par David Temperley dans [18]). Nous avons déjà discuté dans la partie 6 du problème d'utiliser des performances avec notre système. Néanmoins, en segmentant à chaque pulsation notre flux d'entrée (à l'aide d'un marquage réalisé à la main ou par un logiciel spécialisé), nous pourrions limiter les problèmes d'estimation de tempo et nous concentrer sur la quantification. Nous pourrions ensuite apprendre sur ce corpus le jeu de coefficients qui maximise une certaine mesure à déterminer (nombre de transcriptions correctes en $1^{\text{ère}}$ position, nombre de transcriptions correctes dans les n premières positions...)

D'autre part, utiliser une combinaison linéaire des différents critères est une approche assez naïve, rendue né-

cessaire par la contrainte de monotonie. Un moyen de contourner le problème serait de ne plus considérer le poids comme un unique réel, mais plutôt comme un tuple de critères. L'ordre sur le domaine des poids ne serait dans ce cas plus total. On pourrait alors, au lieu d'énumérer les solutions par ordre de poids, énumérer le front de Pareto (*skyline* [5]) et laisser l'utilisateur choisir parmi ces solutions.

8. REFERENCES

[1] Carlos Agon, Gérard Assayag, Joshua Fineberg, and Camilo Rueda. Kant : A critique of pure quantification. In *Proceedings of the International Computer Music Conference*, pages 52–9, 1994.

[2] Carlos Agon, Karim Haddad, and Gérard Assayag. Representation and rendering of rhythm structures. In *International Conference on Web Delivering of Music.*, pages 109–113, 2002.

[3] Miguel Alonso, Bertrand David, and Gaël Richard. Tempo And Beat Estimation Of Musical Signals. In *International Society for Music Information Retrieval Conference (ISMIR)*, 2004.

[4] Gérard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue. Computer-assisted composition at IRCAM : From PatchWork to OpenMusic. *Computer Music Journal*, 23(3) :59–72, 1999.

[5] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The Skyline operator. In *Proceedings of the International Conference on Data Engineering*, pages 421–430, 2001.

[6] Jean Bresson and Carlos Pérez Sancho. New framework for score segmentation and analysis in openmusic. In *Proceedings of the Sound and Music Computing Conference*, 2012.

[7] Ali Taylan Cemgil. *Bayesian Music Transcription*. PhD thesis, Radboud Universiteit Nijmegen, 2004.

[8] Ali Taylan Cemgil, Peter Desain, and Bert Kappen. Rhythm quantization for transcription. *Computer Music Journal*, 24(2) :60–76, 2000.

[9] Peter Desain and Henkjan Honing. The quantization of musical time : A connectionist approach. *Computer Music Journal*, 13(3) :56–66, 1989.

[10] Zoltán Fülöp and Heiko Vogler. Weighted tree automata and tree transducers. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, Monographs in Theoretical Computer Science. An EATCS Series, pages 313–403. Springer Berlin Heidelberg, 2009.

[11] Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics, 2005.

[12] Anssi Klapuri. Musical meter estimation and music transcription. In *Cambridge Music Processing Colloquium*, pages 40–45, 2003.

[13] Stefan Kostka. *Workbook for Tonal Harmony*. McGraw-Hill Education, 1995.

[14] Christopher S. Lee. The Rhythmic Interpretation of Simple Musical Sequences : Towards a Perceptual Model. *Musical Structure and Cognition*, 3 :53–69, 1985.

[15] Benoit Meudic. *Détermination automatique de la pulsation, de la métrique et des motifs musicaux dans des interprétations à tempo variable d'oeuvres polyphoniques*. PhD thesis, Université Pierre et Marie Curie, Paris, 2004.

[16] Declan Murphy. Quantization revisited : a mathematical and computational model. *Journal of Mathematics and Music*, 5(1) :21–34, 2011.

[17] Martin Piszczalski and Bernard A. Galler. Automatic music transcription. *Computer Music Journal*, 1(4) :24–31, 1977.

[18] David Temperley. An evaluation system for metrical models. *Computer Music Journal*, 28(3) :28–44, 2004.

[19] Adrien Ycart. Quantification rythmique dans OpenMusic. Master's thesis, ATIAM – Université Pierre et Marie Curie, Paris, 2015.