Titre: Title:	Ajout de structure aux modèles génératifs de séquences avec la programmation par contraintes
Auteur: Author:	Virasone Manibod
Date:	2022
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Manibod, V. (2022). Ajout de structure aux modèles génératifs de séquences avec la programmation par contraintes [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/10495/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ajout de structure aux modèles génératifs de séquences avec la programmation par contraintes

VIRASONE MANIBOD
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2022

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Ajout de structure aux modèles génératifs de séquences avec la programmation par contraintes

présenté par **Virasone MANIBOD**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Sarath Chandar ANBIL PARTHIPAN, président
Gilles PESANT, membre et directeur de recherche
Claude-Guy QUIMPER, membre externe

REMERCIEMENTS

Je tiens d'abord à remercier mon directeur de recherche Gilles Pesant d'avoir supervisé mon projet de maîtrise. Ce projet ne serait pas possible sans ses idées, sa disponibilité et ses conseils. Je suis reconnaissant pour son encouragement à partager ma recherche. Je voudrais aussi remercier les membres du laboratoire Quosséça qui m'ont accompagné tout au long de mon projet. Je remercie l'acceptation de Sarath Chandar Anbil Parthipan et Claude-Guy Quimper de faire partie des membres du jury afin de valider mes travaux de recherche. Mes remerciements vont également à ma famille et à mes amis pour leur encouragement. Enfin, je suis reconnaissant à IVADO pour avoir financé ma recherche du début jusqu'à la fin.

RÉSUMÉ

Grâce aux avancées de l'apprentissage automatique, il est possible de générer de nouveaux contenus réalistes comme des textes et de la musique. Plusieurs de ces contenus sont modélisés sous la forme d'une séquence, soit une suite ordonnée d'éléments. Cependant, une des difficultés des modèles génératifs de séquences est de manifester une structure dans les exemplaires générés. Par exemple, une phrase ne doit généralement contenir qu'un seul verbe relié à un sujet ou encore, toutes les notes d'une mélodie doivent être en consonance. Ces structures peuvent être requises selon des connaissances préalables ou désirées. Lors de l'entraînement, il n'y a pas de garantie que le modèle puisse efficacement apprendre de telles structures à travers le jeu de données. Ainsi, imposer explicitement la structure par le biais de contraintes pourrait assurer la génération de séquences valides ou souhaitées.

L'objectif du projet est de développer une méthode permettant de contraindre la sortie d'un modèle génératif de séquences. Il est possible d'approcher ce problème lors de l'entraînement ou lors de l'inférence. Dans ce mémoire, nous attaquons le problème lors de la phase d'inférence. En effet, cela permet d'éviter de réentraîner le modèle. Nous faisons aussi nos expériences dans le domaine de la génération de musique, puisque c'est un domaine où le contrôle est toujours un défi actuel. Nous nous basons sur un modèle d'apprentissage automatique représentatif de l'état de l'art, soit le *Chord-conditioned Melody Transformer* (CMT). Ce modèle génère une mélodie selon une suite d'accords donnée. Afin de contraindre les séquences, nous exprimons une structure en utilisant la programmation par contraintes (CP). Cette dernière a l'utilité d'être déclarative et flexible. Par conséquent, il est possible d'exprimer une grande diversité de contraintes. Nous utilisons le solveur MiniCPBP permettant d'effectuer des itérations de *belief propagation* (BP) afin d'approximer la probabilité qu'une valeur d'une variable fasse partie d'une solution réalisable. Afin d'intégrer la distribution de probabilité du CMT dans le processus de BP, la contrainte `oracle` est incluse dans le modèle CP. Cette contrainte associe une probabilité marginale à chaque valeur d'une variable. Elle n'impose donc aucune relation. Par conséquent, à chaque pas de temps, les marginales résultantes de la BP sont représentatives du style appris par CMT et des contraintes du modèle CP.

À travers différentes expériences, diverses contraintes à long terme ont été imposées sur les mélodies (ex., augmentation graduelle de la densité de notes, nombre différent de notes par mesure et présence de toutes les notes de la tonalité). Pour toutes ces contraintes, nous avons fait face au *problème de procrastination* qui consiste à ignorer les contraintes jusqu'au dernier moment possible aboutissant à des mélodies souvent particulières. Pour mitiger le problème,

l'influence du CMT est diminuée lors de la génération de la séquence mélodique. Cependant, parce que l'impact du CMT est réduit, les mélodies deviennent moins similaires au style de musique appris. Par conséquent, nous montrons comment atteindre l'équilibre nécessaire afin de convenablement gérer ces deux enjeux.

La méthode proposée dans ce mémoire est une méthode générale. Bien que nous nous soyons concentrés sur des contraintes à long terme, il est possible d'imposer plusieurs autres types de contraintes. Ainsi, appliquer notre méthode dans le domaine du traitement de la langue naturelle serait une route intéressante pour des travaux futurs.

ABSTRACT

With the recent progress of machine learning, models are now able to produce original realistic content like text or music. However, one popular way of representing these types of content is in a sequential manner. One of the difficulties of sequence generative models in machine learning is that they often struggle to exhibit structure in the generated sequences. For example, a sentence usually needs a predicate and an agent that is performing the action, or a melody needs to stay in the key. Therefore, depending on the domain, such structure can be desired or mandatory to have valid sequences. Relying on the model to learn such structure through the dataset sometimes cannot be reliable because there is no guarantee. For that reason, explicitly imposing constraints may be a solution to the problem.

The objective of this thesis is to develop a method able to constrain the output of a generative sequence model. We tackle the problem during the inference phase where the constraints will guide the generation of the sequences of a pre-trained machine learning model. We also experiment in the domain of music generation where controllability is a common challenge. We build on the Chord-conditioned Melody Transformer (CMT), a state-of-the-art model able to generate a melody based on a given chord progression. The structure to be imposed is expressed as a constraint programming (CP) model. With the high-level modeling and flexibility of CP, many different types of constraints can be enforced. The CP solver used is MiniCPBP which is able to perform belief propagation (BP) to obtain a probability distribution that considers the respect of the constraints. For that distribution to also take into account the music style learned, the probability distribution from CMT is given at each generation step through the `oracle` constraint. That constraint does not enforce any kind of relation between variables. Instead, it associates a marginal probability for each value in the domain of a variable. Thus, the resulting distribution obtained after the process of BP is able to also consider the musical style learned from CMT.

Through different experiments, different long-term constraints were imposed on the melodies such as having increasing number of notes, a different number of notes in each bar and the occurrence of every pitch class in the key. We found that our method has been confronted with the *procrastination problem* when imposing long-term constraints. Indeed, the overall model is almost ignoring the respect of the constraints until the last moment while the sequence is being generated. This phenomenon resulted in very odd melodies. Thus, to mitigate the problem, the impact of the CMT's marginals during BP is reduced. However, this will result in melodies that are less similar than the style learned by CMT. As a consequence, we were

able to achieve the necessary balancing act to decently handle this issue.

Although we focused on long-term constraints in this thesis, our method is general enough to impose other types of constraints. Thus, it would be interesting to apply our approach to other domains of application such as natural language processing.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

BLSTM	LSTM bidirectionnel
BP	*belief propagation*
BPTT	rétropropagation dans le temps
C-RBM	machine de Boltzmann restreinte convolutive
CE	*chord encoder*
CMT	*Chord-conditioned Melody Transformer*
CP	programmation par contraintes
CSP	problème de satisfaction de contraintes
ECE	*expected calibration error*
MSE	erreur quadratique moyenne
EWLD	*Enhanced Wikifonia Leadsheet Dataset*
Hz	Hertz
IA	intelligence artificielle
ILP	programmation linéaire en nombres entiers
JSD	divergence Jensen-Shannon
KLD	divergence de Kullback-Leibler
LSTM	*long short-term memory*
NLL	*negative log likelihood*
NLP	traitement de la langue naturelle
OA	*overlapping area*
PD	*pitch decoder*
RD	*rhythm decoder*
RNN	réseau de neurones récurrent
SRL	étiquetage de rôles sémantiques
VAE	auto-encodeur variationnel

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

Ce mémoire présente un projet de recherche combinant l'apprentissage automatique et la programmation par contraintes afin de structurer la sortie d'un modèle génératif de séquence. Dans ce chapitre, les éléments de la problématique et la motivation seront d'abord abordés. Ensuite, les objectifs seront expliqués. Enfin, le plan du reste du mémoire sera dévoilé.

1.1 Éléments de la problématique

Grâce aux énormes progrès faits durant les dernières décennies, les modèles d'apprentissage automatique sont désormais capables d'accomplir d'impressionnantes tâches. Parmi celles-ci, il y a la génération de nouveaux contenus réalistes. Par exemple, il existe maintenant des modèles ayant la capacité de générer de nouveaux textes [2], images [3] ou musique [4]. Pour ce faire, les modèles ont besoin d'apprendre sur de gros jeux de données maintenant disponibles de nos jours grâce à la numérisation de l'information. Une méthode populaire pour représenter ces contenus est sous la forme d'une séquence [5,6], c'est-à-dire une suite ordonnée d'éléments. Par exemple, un texte peut être représenté par une séquence de mots, une image par une séquence de pixels ou une mélodie par une séquence d'évènements musicaux.

Une des difficultés des modèles génératifs de séquences est qu'il est parfois difficile de manifester une structure [4, 7]. Par exemple, dans une phrase, il doit y avoir un verbe (action) et un agent accomplissant l'action, ou dans une mélodie, les notes doivent faire partie d'une même tonalité. Avoir des séquences permettant de manifester de telles structures permettrait d'améliorer la qualité et le réalisme des séquences générées. Il n'y a pas de garantie que les modèles puissent effectivement capturer les structures présentes dans le jeu de données dans lequel ils sont entraînés [8]. De plus, générer à plusieurs reprises la séquence lors de l'inférence jusqu'à l'obtention d'une séquence structurée peut être coûteux tout en n'ayant, encore une fois, aucune garantie. Ainsi, explicitement structurer la génération de telles séquences par l'entremise de contraintes permettrait de s'assurer qu'une séquence de sortie est valide ou qu'elle respecte certaines règles prédéfinies. Il serait aussi possible d'imposer des contraintes ne se retrouvant pas nécessairement dans le jeu de données d'entraînement. En effet, dans le contexte de la génération de musique, les utilisateurs pourraient, par exemple, contrôler et adapter la sortie du modèle selon leurs préférences.

On peut tenter de résoudre ce problème lors de l'entraînement où le modèle doit apprendre à capturer la structure, par exemple, en modifiant la fonction objectif pour apprendre à

respecter les contraintes [9]. Il est aussi possible d'attaquer le problème lors de la phase d'inférence/génération [6–8, 10]. Effectivement, un modèle est d'abord entraîné sans tenir compte du respect d'une quelconque structure. Par la suite, lors de l'inférence, la séquence doit être guidée vers cette structure. Certaines des techniques lors de l'inférence consistent donc en un type de post-traitement d'une séquence générée.

Dans ce mémoire, nous nous concentrons sur la phase d'inférence. De cette manière, il n'est pas nécessaire de réentraîner le modèle chaque fois pour imposer de nouvelles contraintes (ce qui peut être coûteux). De plus, comme mentionné précédemment, des contraintes qui ne se retrouvent pas nécessairement dans le jeu de données d'entraînement peuvent être imposées. Nous nous concentrons aussi sur le domaine de la génération de musique. En effet, la musique est un domaine très complexe contenant plusieurs dimensions. Plusieurs de celles-ci peuvent être difficiles à capturer par les modèles d'apprentissage automatique (ex., la tonalité, la répétition, la variation ou la structure rythmique) et ce, même pour les modèles faisant partie de l'état de l'art [4]. Également, les architectures traditionnelles de réseaux de neurones n'ont pas été conçues pour permettre du contrôle arbitraire aux sorties générées. Ainsi, dans le contexte de la musique précisément, il serait pertinent de permettre à un utilisateur d'interagir avec les modèles.

1.2 Objectifs de recherche

L'objectif de notre recherche est de développer une méthode permettant de combiner l'apprentissage automatique et la programmation par contraintes afin de contraindre la sortie d'un modèle génératif de séquences. Le modèle d'apprentissage automatique apprendra à imiter le style d'un jeu de données. Afin d'imposer des contraintes aux séquences générées d'un modèle préentraîné, nous utiliserons la programmation par contraintes. En effet, elle est déclarative et flexible, ce qui permettra d'imposer plusieurs différents types de contraintes et de guider la génération vers la structure imposée. La méthodologie comprend aussi de valider notre méthode afin de s'assurer que les séquences générées respectent les contraintes imposées sans toutefois trop s'éloigner du jeu de données appris par le modèle d'apprentissage automatique.

1.3 Plan du mémoire

Dans le chapitre 2, nous présentons les concepts de base afin de comprendre le domaine de recherche. Par la suite, au chapitre 3, une revue de littérature est présentée sur les différentes façons d'ajouter une structure à la sortie d'un modèle d'apprentissage automatique. Ensuite,

au chapitre 4, nous expliquons notre méthode pour combiner l'apprentissage automatique et la programmation par contraintes pour imposer une structure aux séquences de sortie. Puis, au chapitre 5, nous montrons nos expériences pour tester notre méthode. Ce chapitre présente aussi les différents problèmes auxquels nous avons fait face, et comment nous les avons gérés. Enfin, au chapitre 6, nous concluons le mémoire en discutant des limitations, des améliorations et des travaux futurs.

CHAPITRE 2 CONCEPTS DE BASE

Ce chapitre présente les concepts de base nécessaires à la compréhension du domaine de recherche du mémoire. Tout d'abord, nous dériverons les principaux algorithmes d'apprentissage automatique concernant la génération de séquences. Ensuite, nous expliquerons la programmation par contraintes. Enfin, un bref résumé des concepts de base en théorie de la musique sera présenté.

2.1 Apprentissage automatique

L'apprentissage automatique est une branche de l'intelligence artificielle (IA) [11]. Il consiste à utiliser des algorithmes sur un jeu de données afin qu'un modèle puisse apprendre des motifs ou des régularités découlant de ces exemplaires. Le modèle peut ensuite appliquer ces motifs afin d'accomplir des tâches telles que la prédiction, la classification, le partitionnement de données et la génération. Un des points attrayants de cette branche de l'IA est la capacité des machines à rapidement analyser plusieurs milliers de fichiers de données contrairement à un humain où il serait presque impossible de le faire manuellement. Il existe trois types principaux de l'apprentissage automatique : l'apprentissage supervisé, non supervisé et par renforcement. Cependant, dans ce mémoire, seulement l'apprentissage supervisé est pertinent. Ainsi, ce dernier consiste à avoir un jeu de données où chaque exemplaire possède une étiquette (ou une cible). Le but est alors de pouvoir prédire l'étiquette d'un nouvel exemplaire.

2.1.1 Entraînement

Un modèle d'apprentissage automatique est entraîné semblablement à un humain. Autrement dit, le modèle apprend de ses erreurs sur plusieurs itérations. Pour la plupart des modèles, des paramètres θ doivent être graduellement ajustés afin de trouver les valeurs donnant la meilleure performance possible. Plus précisément, on cherchera à trouver les valeurs des paramètres minimisant une fonction de coût $J(\theta)$ (ou fonction de perte) représentant la différence (l'erreur) entre les valeurs produites par le modèle et les valeurs cibles. Un exemple de fonction de coût est l'erreur quadratique moyenne (MSE) :

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - h_\theta(x_i))^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

où n est le nombre d'exemplaires dans le jeu de données, y_i la valeur cible de l'exemplaire i et $h_\theta(x_i)$ est la prédiction faite par le modèle h pour les données en entrée x_i de l'exemplaire i selon les paramètres θ . La notation de $h_\theta(x_i)$ est simplifiée par $\hat{y}_i$.

2.1.2 Descente du gradient

Selon la complexité du modèle, trouver les paramètres permettant d'obtenir le minimum global de la fonction de coût n'est pas une simple tâche. De ce fait, une méthode très populaire pour trouver un minimum local d'une fonction est l'algorithme de la descente du gradient. Pour utiliser cet algorithme, il est d'abord primordial que la fonction de coût $J(\theta)$ soit différentiable. La descente du gradient consiste à calculer la pente la plus abrupte (gradient) de la fonction $J(\theta)$ selon chaque paramètre par rapport à un point donné. Ainsi, l'intuition est de descendre localement d'un certain pas vers la direction opposée du gradient. En réalisant ces pas de façon itérative, il est possible de converger vers un minimum local.

Par conséquent, au début de l'entraînement d'un modèle, les paramètres sont initialisés par des valeurs au hasard ou selon une heuristique. Par la suite, le modèle génère des prédictions pour chacun des exemplaires à partir de ses paramètres courants. Le gradient de la fonction de coût est ensuite calculé par rapport aux erreurs des prédictions faites. Les paramètres sont ainsi mis à jour de la manière suivante :

$$\theta = \theta - \eta \nabla J(\theta) \tag{2.2}$$

où η est le taux d'apprentissage, soit la grandeur du pas vers la direction opposée du gradient. Ce processus est répété jusqu'à l'atteinte d'un critère d'arrêt (ex., un nombre d'itérations maximal).

2.1.3 Réseau de neurones

Un réseau de neurones est un type de modèle où un ensemble de nœuds, appelé neurones, sont connectés entre eux par des poids (paramètres) afin de modéliser des fonctions complexes non linéaires. Un neurone (figure 2.1) est une unité de calcul où une combinaison linéaire avec ses poids $\boldsymbol{w}$ (vecteur colonne) est faite avec les données en entrée $\boldsymbol{x}$ (un vecteur colonne). Une fonction d'activation σ est ensuite appliquée sur le résultat de la combinaison linéaire :

$$\sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{2.3}$$

où b est le biais. Des exemples de fonction d'activation σ sont la sigmoïde et la ReLU :

$$\sigma(z) = \text{sigmoïde}(z) = \frac{1}{1 + \mathrm{e}^{-z}} \tag{2.4}$$

$$\sigma(z) = \text{ReLU}(z) = \max(0, z) \tag{2.5}$$

C'est grâce à ces fonctions d'activation que les réseaux de neurones ont la capacité de capturer des relations non linéaires.

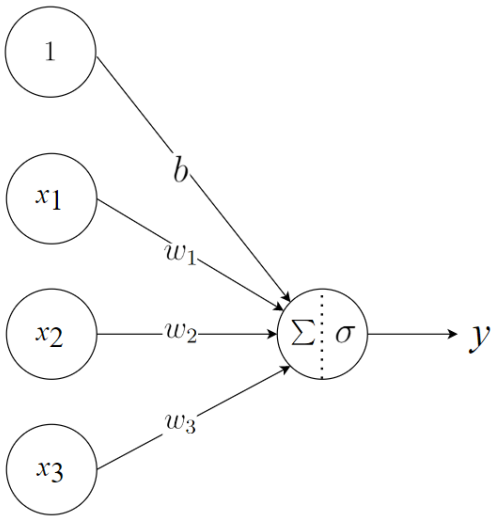

Figure 2.1 Neurone d'un réseau de neurones.

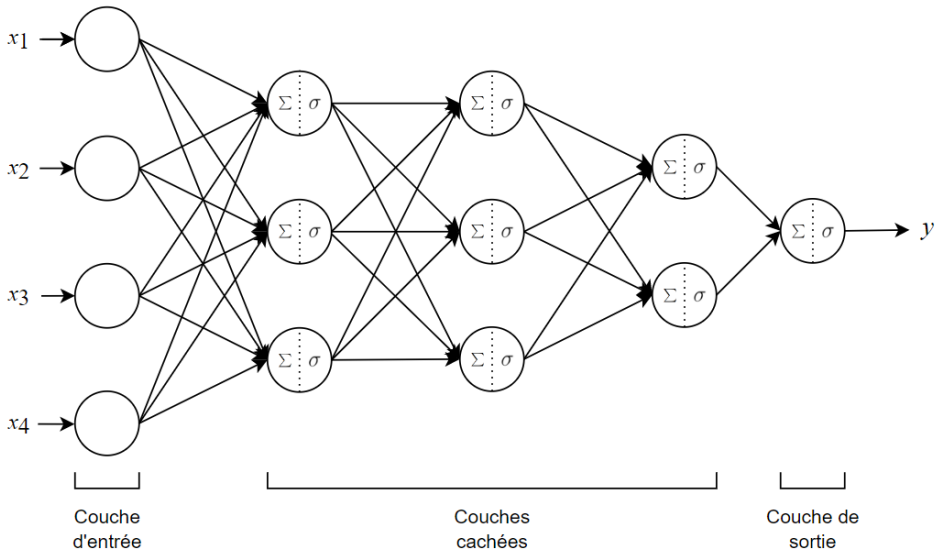

Figure 2.2 Exemple d'une architecture d'un réseau de neurones.

Pour continuer, l'architecture d'un réseau est composée d'une couche d'entrée, de sortie et de couches cachées (figure 2.2). Entraîner un réseau de neurones est similaire à entraîner un simple modèle d'apprentissage automatique. Les poids de chaque neurone sont initialisés au hasard ou selon une heuristique. Ensuite, la propagation avant est faite. Elle consiste à donner à la couche d'entrée les données d'entrée. Les calculs des neurones se propagent donc à travers le réseau jusqu'à l'obtention des valeurs de la couche de sortie. L'erreur entre la sortie obtenue et la sortie cible est ensuite calculée. Par la rétropropagation, le calcul du gradient du réseau est fait, et les poids des connexions des neurones sont ajustés. Ainsi, la descente du gradient peut être appliquée afin de minimiser la fonction de coût.

2.1.4 Réseau de neurones récurrent

Les simples réseaux de neurones présentés à la section précédente permettent seulement d'admettre des données de taille fixe comme entrée et sortie. Un réseau de neurones récurrent (RNN) est un type de réseau permettant de gérer des données séquentielles à taille variable. Des exemples de données séquentiels sont les textes et les mélodies. En effet, ils peuvent être représentés par une séquence où chaque élément (jeton) correspond à un mot et une note respectivement. Tout comme avec un réseau de neurones, un RNN est composé d'une couche d'entrée, de sortie et de couches cachées. Cependant, un RNN contient aussi un état caché h (figure 2.3). L'état caché sert de mémoire pour les éléments de la séquence précédemment vus. À chaque pas de temps t, l'état caché est mis à jour et réutilisé comme entrée pour le prochaine pas de temps. Pour entraîner un tel modèle, celui-ci essaie de prédire le prochain jeton x_{t+1} selon le jeton courant x_t et passés.

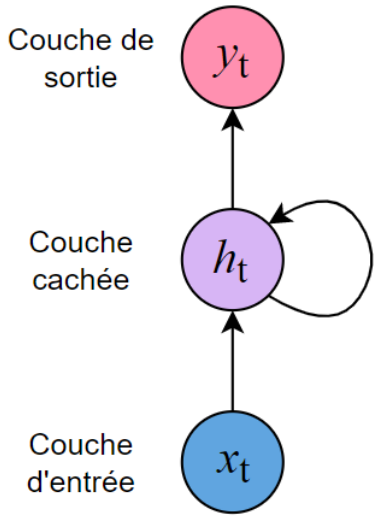

Figure 2.3 Exemple d'une architecture d'un réseau de neurones récurrent.

Par conséquent, en entrée, un RNN reçoit le jeton courant x_t de la séquence et l'état caché du pas de temps précédent h_{t-1} afin de prédire le prochain jeton. La méthode utilisée pour calculer le gradient dans les réseaux de neurones récurrents est une méthode adaptée de la rétropropagation, soit la rétropropagation dans le temps (BPTT). Une fois entraîné, un RNN peut être utilisé afin de générer de nouvelles séquences.

2.1.5 LSTM

Le RNN est dit être un modèle à mémoire à court terme. En effet, il n'est pas très efficace pour retenir les éléments passés sur plusieurs pas de temps. Ce problème est causé par le phénomène du *vanishing gradient* ou du *exploding gradient* lors de la BPTT. Effectivement, pendant l'entraînement, les multiplications successives afin de calculer le gradient ont la possibilité de réduire le gradient à une valeur presque nulle. De ce fait, comme l'ajustement d'un paramètre est proportionnel au gradient, les poids ne changent pratiquement pas. Ainsi, le *Long Short-Term Memory* (LSTM) [12] arrive à considérablement résoudre ce problème. Ce dernier est composé de cellules de mémoire (figure 2.4). L'état caché h et un état de cellule C représentant la mémoire à long terme sont transférés d'une cellule à l'autre. Par l'entremise de portes, l'état de cellule peut être modifié dépendamment de quelles informations devraient être oubliées, ajoutées ou sorties. De cette manière, le modèle est en mesure de mieux gérer sa mémoire et de générer des séquences avec de meilleures dépendances à long terme que le RNN original.

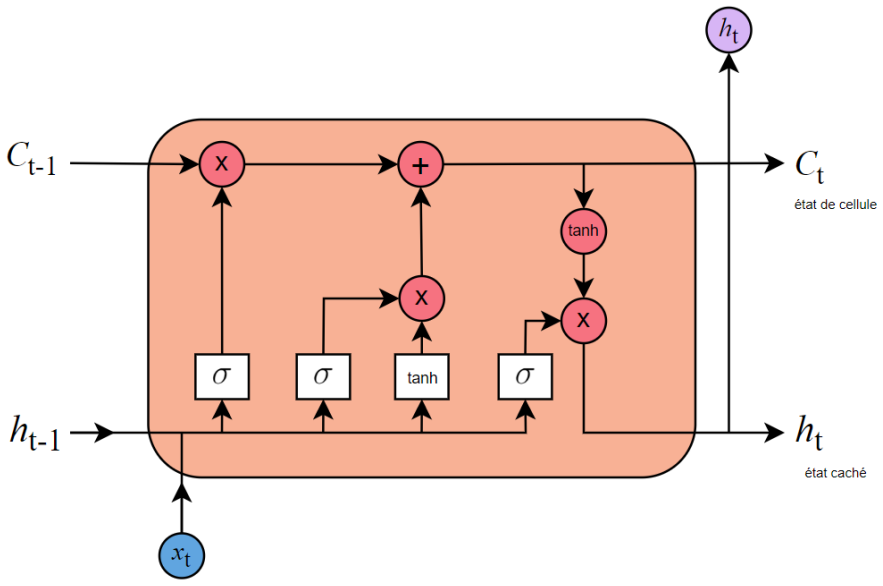

Figure 2.4 Cellule d'un LSTM.

2.1.6 Mécanisme de l'attention

Un autre moyen de capturer les dépendances à long terme pour les séquences est le mécanisme de l'attention [13]. Ce mécanisme permet d'indiquer à chaque pas de temps les éléments de la séquence d'entrée auxquels le modèle devrait porter le plus attention afin de mieux décoder le jeton courant en sortie. L'attention est obtenue en faisant une somme pondérée de tous les états cachés de l'encodeur ($h_1 \dots h_T$) afin d'obtenir un vecteur de contexte (figure 2.5). La pondération de chaque état caché est déterminée par $\alpha_{i,j}$ signifiant à quel point le modèle devrait porter attention à l'élément x_j afin de décoder le jeton y_i. Le jeton y_t courant est ensuite généré à l'aide de ce vecteur de contexte et de l'état caché du décodeur s_{t-1}. Le Transformer [14], un modèle ayant eu beaucoup de succès dans le domaine du traitement de la langue naturelle (NLP), utilise l'auto-attention [15]. Ce mécanisme analyse, à la place, les relations entre les éléments d'une même séquence. Ainsi, lors de la génération de la séquence, cette dernière peut faire référence à des éléments passés.

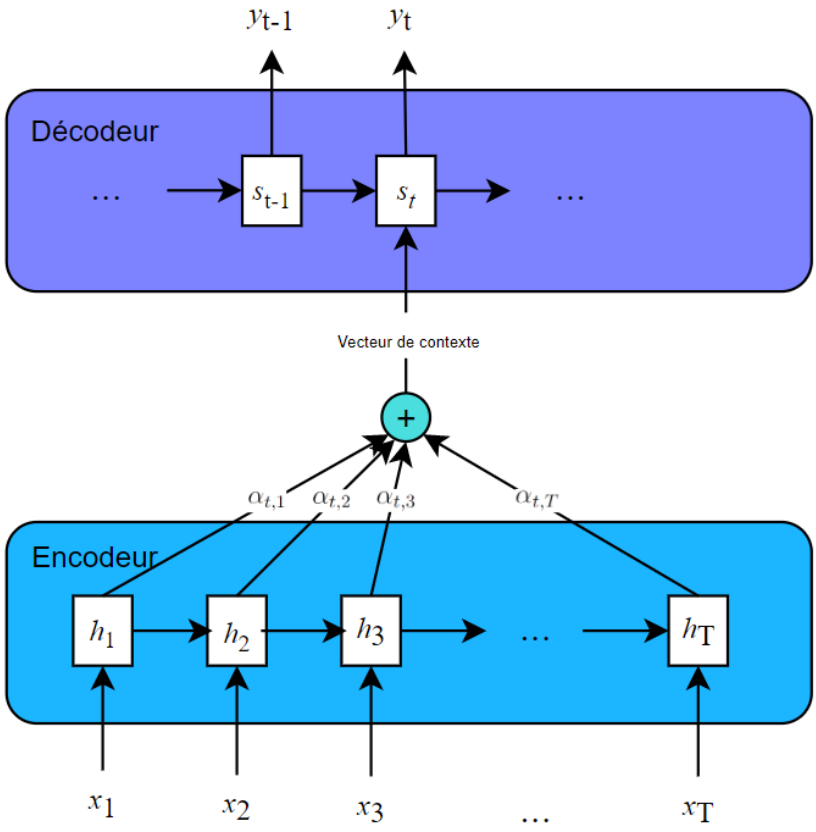

Figure 2.5 Visualisation du mécanisme de l'attention.

2.2 Programmation par contraintes

La programmation par contraintes (CP) [16] est un paradigme de programmation déclarative servant à résoudre des problèmes combinatoires. La popularité de la CP repose sur la description des caractéristiques d'un problème à travers des contraintes de haut niveau sans toutefois spécifier comment le résoudre. Certaines de ces contraintes peuvent être posées par des contraintes dites globales permettant d'encapsuler des relations non linéaires. La résolution du problème se fait ensuite à l'aide d'un solveur CP où des algorithmes et des mécanismes sont implémentés afin de rechercher une solution au problème. Ainsi, la CP peut être divisée en deux parties : la modélisation et la recherche. Dans les prochaines lignes, ces deux parties seront expliquées un peu plus en détail.

2.2.1 Modélisation

La modélisation consiste à décrire le problème à résoudre à l'aide de contraintes. Dans ce mémoire, nous nous concentrons sur la résolution de problèmes de satisfaction de contraintes (CSP). Un CSP est tuple (X, D, C) où $X = \{x_1, \ldots, x_n\}$ est un ensemble fini de variables, $D = \{d_1, \ldots, d_n\}$ est un ensemble fini de domaines contenant des valeurs et $C = \{c_1, \ldots, c_m\}$ est un ensemble fini de contraintes. Chaque variable x_i peut être assignée à une valeur parmi celles de son domaine $D(x_i) = d_i$ respectif. Chaque contrainte c_i représente une relation entre des variables d'un sous-ensemble de X.

Comme exemple de problème combinatoire, prenons le carré latin. Le problème consiste à remplir une grille l x l avec l symboles différents. Chaque symbole doit apparaître exactement une fois dans chaque ligne et colonne. La figure 2.6 montre une solution possible d'une grille 4 x 4. Le problème pour ce carré latin peut être modélisé par un CSP de la façon suivante :

$$X = \{x_1, \ldots, x_{16}\} \tag{2.6}$$

$$D = \{d_i = \{0, \ldots, 3\} \quad \forall i \in \{1, \ldots, 16\}\} \tag{2.7}$$

$$\begin{aligned} C = \{&\texttt{alldifferent}(x_{4i+1}, x_{4i+2}, x_{4i+3}, x_{4i+4}) \quad \forall i \in \{0, \ldots, 3\}, \\ &\texttt{alldifferent}(x_i, x_{i+4}, x_{i+8}, x_{i+12}) \qquad \forall i \in \{1, \ldots, 4\}\} \end{aligned} \tag{2.8}$$

Il y a 16 variables afin de représenter les 16 (4x4) cases de la grille. Chaque case a la possibilité d'être affectée à un des 4 symboles. Ainsi, leur domaine correspond au chiffre de 0 à 3. Ensuite, la contrainte globale `alldifferent` permet de spécifier que les variables dans l'ensemble donné doivent être affectées à des valeurs différentes. Par conséquent, la contrainte `alldifferent` est déclarée pour chaque ensemble de variables correspondant à une ligne ou

3	0	1	2
2	3	0	1
1	2	3	0
0	1	2	3

Figure 2.6 Exemple d'un carré latin 4x4 résolu.

une colonne. Il y a donc 8 contraintes `alldifferent` déclarées au total.

2.2.2 Filtrage et Propagation

Une contrainte peut être considérée comme un algorithme qui filtre les valeurs incohérentes des domaines des variables impliquées, soit celles ne pouvant pas localement mener à une solution possible. La réduction de domaine d'une variable est communiquée à toutes les autres contraintes impliquant cette variable. C'est ce qu'on appelle la propagation de contraintes. Ces autres contraintes peuvent ainsi filtrer d'autres valeurs devenues incohérentes localement, et à leur tour, propager ces changements. Ce processus de propagation est garanti qu'il finit, car les domaines des variables sont finis et ne peuvent que diminuer.

2.2.3 Intérêt des contraintes globales

Il est souvent possible de modéliser un même problème en utilisant différentes variables, domaines ou contraintes. Ces choix peuvent grandement affecter l'efficacité de la résolution du problème. En effet, pour l'exemple du carré latin, il est aussi possible d'utiliser des inégalités binaires au lieu de la contrainte globale `alldifferent`. Par exemple, pour la première ligne (variables x_1, x_2, x_3, x_4) les contraintes $x_1 \neq x_2,\ x_1 \neq x_3,\ x_1 \neq x_4,\ x_2 \neq x_3,\ x_2 \neq x_4,\ x_3 \neq x_4$ sont logiquement équivalentes. En revanche, six fois plus de contraintes sont nécessaires. De plus, chaque contrainte est binaire. Ainsi, elles ne peuvent que filtrer localement les valeurs de deux variables. Par conséquent, plus d'itérations de propagation seraient nécessaires afin de filtrer toutes les valeurs incohérentes. D'un autre côté, la contrainte `alldifferent` a une vue plus globale sur toutes les variables impliquées, et permet d'être plus efficace.

2.2.4 Recherche

Pour continuer, le filtrage des valeurs des domaines n'est souvent pas suffisant pour arriver à une solution au problème. Par conséquent, pour continuer la recherche, un branchement est fait. Plus précisément, une des variables est affectée à une valeur de son domaine courant. Le changement du domaine de cette variable se propage aux contraintes engendrant possiblement le retrait de valeurs incompatibles pour d'autres variables. L'alternance entre les filtrages des domaines et les branchements se font jusqu'à l'obtention d'une solution, c'est-à-dire qu'il ne reste qu'une valeur dans chaque domaine des variables. Pour continuer, le branchement peut parfois causer qu'un domaine d'une variable devienne vide à cause des filtrages engendrés. Autrement dit, aucune solution n'est possible par cette affection. Dans ce cas, une marche arrière (*backtracking)* est exécutée, et un autre choix de variable et de valeur de branchement est fait. Les choix variable et valeur peuvent être déterminés à l'avance par l'utilisateur. Ces choix d'heuristique de branchement peuvent fortement influencer le temps de résolution du problème.

2.3 Théorie de la musique

Les concepts de base en musique seront expliqués dans la notation du solfège, car c'est la notation utilisée en français. Cependant, notez que la notation ABC est le système majoritairement utilisé dans la littérature.

Une note est définie par sa hauteur de note et sa durée. Une hauteur de note correspond à une fréquence en Hertz (Hz). Plus la fréquence est élevée, plus la note est aiguë. Cependant, les notes sont plutôt identifiées par des syllabes plutôt que par leur fréquence (ex., *la* pour une fréquence de 440 Hz). Il y a 12 classes de note différentes, soit *do*, $do^\sharp(ré^\flat)$, *ré*, $ré^\sharp(mi^\flat)$, *mi*, *fa*, $fa^\sharp(sol^\flat)$, *sol*, $sol^\sharp(la^\flat)$, *la*, $la^\sharp(si^\flat)$ et *si*. Ces notes se répètent à travers les fréquences du spectre audio. Par exemple, les fréquences 220 Hz et 440 Hz correspondent toutes les deux à un *la*. Cependant, l'une sera plus aiguë que l'autre. Il est possible d'ajouter un chiffre indiquant l'octave (la fréquence) précise de la note pour différencier deux mêmes classes de hauteur de note (ex., la^2 vs la^3). La durée d'une note indique combien de temps la hauteur de note est maintenue (ex., la ronde, la blanche et la noire correspondante à 4, 2 et 1 temps respectivement). Un silence signifie qu'aucun son n'est joué. Il est défini par sa durée (ex., pause, demi-pause et soupir correspondant à 4, 2, et 1 temps respectivement) indiquant combien de temps le silence est maintenu.

Un intervalle entre deux notes est l'écart de leur hauteur de note. Un intervalle harmonique est l'écart entre deux notes jouées en même temps, alors que l'intervalle mélodique est l'écart

entre deux notes successives (voir figure 2.7). L'intervalle le plus petit est le demi-ton. Par exemple, chaque paire de notes successives parmi les 12 classes de note mentionnées précédemment a un intervalle d'un demi-ton. Un intervalle est généralement identifié par un nom, par exemple, tierce mineure (3 demi-tons), tierce majeure (4 demi-tons), quinte (7 demi-tons) ou octave (12 demi-tons). Particulièrement, l'octave représente une fréquence deux fois plus élevée ou une distance entre deux mêmes classes de hauteur de note consécutives. Certains de ces intervalles sont plus en consonance, alors que d'autres sont plus en dissonance.

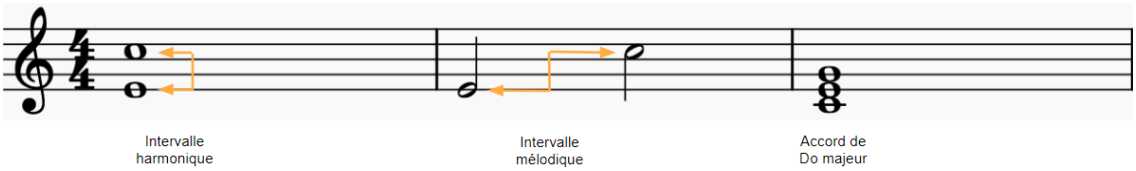

Figure 2.7 Différence entre un intervalle harmonique (gauche) et un intervalle mélodique (centre). Exemple d'un accord de *do* majeur (droite).

Un accord est un ensemble d'au moins trois notes jouées simultanément. Chaque accord contient une note fondamentale sur laquelle sera construit l'accord. Les intervalles des autres notes par rapport à la note fondamentale permettront de caractériser le type d'accord. Par exemple, l'accord du *do* majeur est composé des notes *do* (la fondamentale), *mi* (à une tierce majeure de *do*) et *sol* (à une quinte de *do*). Un accord reste le même type d'accord peu importe l'octave des notes qui le constituent. Par conséquent, il y a différentes façons de jouer un même accord.

Le rythme représente la structure temporelle de la musique. Ceci inclut la pulsation, la durée des notes et l'accentuation à certains temps. Une musique est normalement divisée en sections égales appelées mesures (voir figure 2.8). Dans une partition, chaque mesure est séparée par des barres verticales (barres de mesure). La signature de temps est une fraction en début de partition permettant de donner des informations sur la structure rythmique d'une pièce de musique. Le numérateur permet de donner le nombre de temps que contient une mesure. Le dénominateur représente la durée d'un temps (en fraction de la ronde). Par exemple, la signature de temps 3/4 indique qu'il y a 3 temps dans une mesure, et que chaque temps équivaut à une noire (car la noire vaut 1/4 fois la ronde). Différentes signatures de temps auront différentes accentuations à des temps spécifiques.

Simplement, une tonalité est un ensemble de classes de hauteurs de note qui sonnent bien ensemble. Une gamme est cet ensemble de notes placé dans un certain ordre précis. Une tonalité est caractérisée par sa tonique et son mode. La tonique est la première note de la gamme avec laquelle les autres notes seront déterminées. La tonique permet aussi de faire

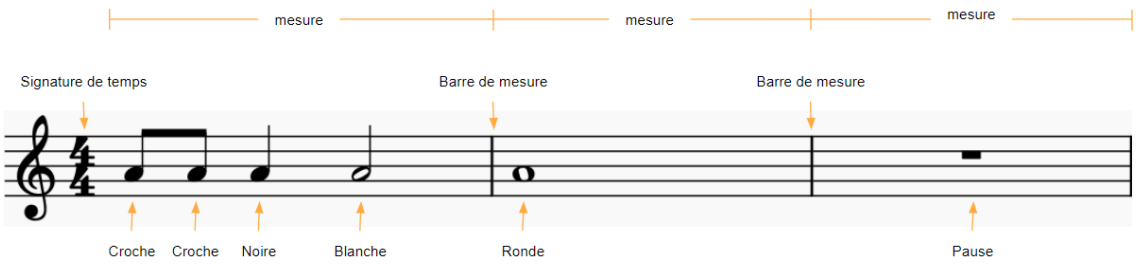

Figure 2.8 Concepts rythmiques de base en musique.

ressentir un sentiment de résolution lorsque jouée. Le mode est déterminé par la séquence d'intervalles entre chaque note successive de la gamme. Il existe plusieurs modes de tonalité, mais les deux plus fréquents sont le mode majeur et mineur. Par exemple, la gamme de *do* majeur est composée des notes suivantes : *do*, *ré*, *mi*, *fa*, *sol*, *la*, et *si* (les touches blanches d'un piano). Une musique dans la tonalité de *do* majeur utiliserait les notes de la gamme de *do* majeur. En général, une musique est composée en utilisant seulement les notes faisant partie d'une tonalité. Cependant, il peut être aussi très commun d'utiliser des notes hors de la tonalité et d'utiliser plus d'une tonalité dans une même musique. Différentes tonalités font ressentir différentes émotions.

Pour continuer, en général, une mélodie est une séquence de notes jouées l'une à la suite de l'autre. La plupart du temps, dans une chanson, la mélodie est la partie la plus mémorable. La transposition d'une mélodie consiste à changer toutes les hauteurs de note d'un intervalle constant. Par exemple, en transposant les notes *do* et *mi* de 5 demi-tons, celles-ci deviennent *fa* et *la*.

CHAPITRE 3 REVUE DE LITTÉRATURE

Dans ce chapitre, nous présentons les différents travaux essayant d'imposer une structure ou des contraintes à la sortie d'un modèle d'apprentissage automatique. Nous verrons diverses méthodes appliquées lors de la phase d'entraînement ou d'inférence. Dans cette section, nous nous intéressons aux techniques appliquées dans le domaine de la génération de musique, mais aussi dans le domaine du traitement de la langue naturelle. En effet, plusieurs travaux dans ce domaine peuvent être considérés comme pertinents et similaires à notre méthode proposée.

3.1 Structure imposée à l'entraînement

Google Magenta [17] est un projet dans le but d'étudier les limites des modèles d'apprentissage automatique en tant qu'outils créatifs permettant de participer au processus de création des humains. Magenta englobe, entre autres, les domaines de la génération d'image et de musique. Parmi leurs modèles dans le domaine musical, on trouve le Lookback RNN et l'Attention RNN. Ces deux modèles permettent d'améliorer la structure à long terme des RNNs. Avec le Lookback RNN, des informations additionnelles sur les évènements passés sont données en entrée du modèle. Dans le contexte de la musique, ceci inclut les évènements des deux dernières mesures, si le dernier évènement est répété parmi les deux dernières mesures et la position au sein de la mesure. Aussi, deux nouvelles étiquettes ont été créées : répéter l'évènement se retrouvant une mesure plus tôt et répéter l'évènement se retrouvant deux mesures plus tôt. Par conséquent, avec les entrées et étiquettes additionnelles, le Lookback RNN est capable de capturer les répétitions de motifs musicaux des deux dernières mesures. D'une autre part, l'Attention RNN utilise un masque d'attention sur les n derniers évènements générés. Il peut ainsi savoir sur quels évènements passés le modèle devrait porter le plus d'attention. Dès lors, ces deux modèles sont capables de répéter des motifs se trouvant dans les dernières mesures en donnant explicitement en entrée des informations habilement choisies sur les évènements passés.

Le Transformer par Vaswani *et al.* [14] a eu beaucoup de succès dans le domaine du NLP. Il peut aussi être utilisé dans le domaine de la musique si cette dernière est considérée comme une séquence d'évènements musicaux. Le Transformer se base sur l'auto-attention en utilisant un encodage absolu des positions des éléments dans la séquence. Cependant, en musique, la position relative est aussi très importante, voire même plus importante (ex., temps ou intervalle entre deux notes). Shaw *et al.* [18] ont proposé une façon de mieux incorporer les

distances relatives dans l'équation de l'auto-attention en y ajoutant un nouveau terme à calculer. Pour calculer l'auto-attention relative, l'instance d'un tenseur R intermédiaire de dimension (L,L,D) est nécessaire où L est la longueur de séquence et D la dimension de l'état caché du modèle. La complexité en espace pour calculer l'auto-attention relative est donc $O(L^2D)$. Le Music Transformer [5] propose, par la suite, une implémentation efficace en mémoire de l'auto-attention relative. En effet, les auteurs ont trouvé une façon d'éviter d'instancier R. Ils utilisent une technique qu'ils appellent le "skewing trick". Cette technique consiste à manipuler (ex., *padding*, *reshaping*, *slicing*) les tenseurs déjà instanciés afin d'obtenir le même résultat sans instancier R. De ce fait, la complexité en espace devient $O(LD)$. Par conséquent, il est possible d'entraîner le modèle sur des séquences de musique plus longue, tout en capturant les relations relatives entre chaque paire d'évènements. Ainsi, le Music Transformer est capable de générer des musiques avec des motifs répétés et variés même au-delà de la longueur de séquence avec laquelle le modèle a été entraîné.

Malgré les bonnes performances du Music Transformer à continuer un motif musical, celui-ci ne permet pas à un utilisateur de contrôler sa sortie. Ainsi, Young *et al.* [19] propose une méthode afin d'améliorer la contrôlabilité du Music Transformer. La méthode repose sur l'utilisation du *Bias-tuning* [20]. Elle consiste à ajuster (*fine-tuning*) les paramètres du Music Transformer afin de lui apprendre à générer des morceaux musicaux manifestant les contraintes désirées. En général, l'ajustement des paramètres peut être coûteux dépendamment du nombre de contraintes imposées. Cependant, le *Bias-tuning* n'ajuste qu'une partie des paramètres du modèle. Plus précisément, seulement les biais sont ajustés. De ce fait, il devient moins coûteux d'ajuster un modèle. Pour continuer, les auteurs proposent aussi une fonction de perte contrastive essayant de trouver la contrainte permettant de mieux "expliquer" un exemplaire du jeu de données. Avec ces techniques, il devient beaucoup plus probable pour le Music Transformer de générer une continuation musicale manifestant les contraintes imposées. Par exemple, en essayant d'imposer 6 contraintes à la fois, dans 8% des cas, le modèle arrive à générer une sortie qui les respecte. Avec le modèle original du Music Transformer, cela est arrivé dans 0% des cas. Les auteurs ont aussi été en mesure d'ajouter plusieurs différents types de contrôle de haut niveau. Un test d'écoute montre également que leur méthode n'empiète pas sur la qualité de la musique générée. Enfin, la limitation semble être la garantie du respect des contraintes imposées. En effet, afin d'obtenir une sortie valide, les auteurs ont recours à la méthode du rejet (*rejection sampling*).

Hadjeres et Nielsen proposent l'Anticipation-RNN [21], un modèle capable de générer une mélodie avec la possibilité d'imposer des contraintes sur les notes. En d'autres mots, les contraintes consistent à spécifier des hauteurs de note à des pas de temps données. L'approche naïve consistant à simplement forcer une note à un temps précis viendrait perturber

la distribution des notes générées précédemment. Ainsi, l'Anticipation-RNN est composé de deux RNNs, soit le Token-RNN et le Constraint-RNN. Ce dernier parcourt une séquence de droite à gauche et rassemble les futures contraintes se trouvant entre le pas de temps t courant et le dernier pas de temps T. Le Token-RNN est un modèle générant les jetons de la mélodie de façon autorégressive. La sortie du Constraint-RNN est concaténée à l'entrée du Token-RNN. Par conséquent, ce dernier est en mesure de générer les notes de la mélodie en tenant compte des contraintes qui arrivent. La distribution des notes est ainsi plus cohérente qu'une approche naïve. Une limitation de l'Anticipation-RNN est qu'il est seulement possible d'imposer un type spécifique de contrainte.

Le MusicVAE par Roberts *et al.* [22] se base sur l'architecture d'encodeur-décodeur typique d'un auto-encodeur variationnel (VAE) [23]. L'encodeur utilisé est un LSTM bidirectionnel (BLSTM). Le décodeur est composé d'un *conductor* RNN et d'un *bottom-layer* RNN. Plus concrètement, la séquence d'entrée x est divisée en sous-séquences sans chevauchement et encodée par le BLSTM pour obtenir un vecteur latent z. Avec le vecteur latent z comme état caché initial, le *conductor* RNN génère un plongement pour chacune de ces sous-séquences. Ensuite, le *bottom-layer* RNN génère les jetons de chaque sous-séquence de façon autorégressive. Lors de la génération d'une nouvelle sous-séquence, l'état initial du *bottom-layer* RNN est réinitialisé avec le plongement correspondant généré par le *conductor* RNN. De cette manière, seulement les plongements, qui représentent l'espace latent, peuvent être utilisés par le modèle pour capturer une structure à long terme. En d'autres mots, l'espace latent est plus habilement utilisé. En effet, avec un simple RNN en tant que décodeur, lorsque l'état caché initial est initialisé avec l'espace latent, son effet devient moins important plus la séquence est générée. De ce fait, la sortie du modèle devient presque indépendante de l'espace latent z. C'est le "posterior collapse", un problème courant pour les modèles VAE. Ainsi, le MusicVAE est capable de produire des séquences avec une meilleure structure à long terme que les autres modèles se basant sur l'architecture d'un VAE. Le modèle permet aussi de manipuler l'espace latent afin de contrôler certaines caractéristiques comme la densité des notes, l'intervalle moyen ou la présence des notes de la gamme.

3.2 Structure imposée à l'inférence

Dans [10], Pachet *et al.* utilise un modèle de Markov M afin d'apprendre à imiter le style d'un jeu de données de musique. Les contraintes sont formulées sous la forme d'un CSP afin de filtrer les transitions rendues impossibles. Les transitions filtrées auront leur probabilité fixée à zéro, alors que les transitions restantes auront leur probabilité normalisée. Le modèle résultant est un modèle de Markov non homogène $\tilde{M}$, c'est-à-dire que la matrice de transition change à

travers le temps. Ainsi, $\tilde{M}$ peut seulement générer des solutions qui respectent les contraintes imposées. De plus, grâce à leur processus de normalisation, $\tilde{M}$ suit les mêmes probabilités de transition que le modèle original M. Pour générer une solution, il suffit d'effectuer une marche aléatoire sur $\tilde{M}$. Une limitation de cette méthode est que les contraintes doivent se trouver dans la portée/ordre du modèle Markov qui est normalement 1. Cela inclut les contraintes unaires et les contraintes binaires entre deux éléments voisins. Aussi, plus l'ordre du modèle est élevé, plus il est dispendieux de l'entraîner. Par conséquent, pour le long terme, les modèles de Markov ne sont pas appropriés.

Une machine de Boltzmann restreinte [24] convolutive (C-RBM) est utilisée par Lattner *et al.* comme modèle génératif [7]. Lors de l'étape de génération, une musique de référence est employée afin d'imposer des contraintes comme la structure d'auto-similarité, la tonalité et le rythme. En d'autres mots, la musique de référence est utilisée comme un gabarit par rapport aux contraintes. Ces dernières sont exprimées sous la forme de fonctions de coût différentiables. Chaque fonction de coût représente la différence (l'erreur) entre la structure de l'échantillon et le gabarit. Ainsi, en utilisant la descente de gradient, ces fonctions sont minimisées. Plus précisément, la méthode utilisée est l'échantillonnage sous contrainte. Tout d'abord, un échantillon est aléatoirement initialisé par une distribution uniforme. Ensuite, des itérations de descente de gradient et d'échantillonnage de Gibbs sont effectuées. La descente de gradient permet d'imposer les contraintes, alors que l'échantillonnage de Gibbs permet de se rapprocher du style de musique appris par le modèle lors de l'entraînement. Ces itérations sont faites jusqu'à la convergence. La métaheuristique du recuit simulé est aussi employée afin d'échapper aux optimums locaux et pour guider l'exploration dans l'espace de recherche. Les échantillons de musique sortant de ce modèle réussissent à atteindre une structure de haut niveau par l'imposition de contraintes molles. Le contrôle de la génération se fait par le choix de la musique de référence. Les auteurs mentionnent comme limitation qu'il est seulement possible d'imposer des contraintes de haut niveau, sauf si le modèle est entraîné sur un petit jeu de données. De plus, seulement trois types de contraintes sont imposées. En effet, plusieurs autres types de contraintes sont souhaitables comme la transition entre les différentes sections d'une musique.

Dans le domaine du NLP, une tâche populaire où ajouter des contraintes à la génération d'une séquence peut s'avérer utile est l'étiquetage de rôles sémantiques (SRL). La tâche consiste à identifier les rôles sémantiques des différents mots (ou groupes de mots) d'une phrase selon un prédicat. Par exemple, dans la phrase *Le chien court dans la rue.*, le prédicat est le mot *court*. Il faut ainsi prédire le rôle des arguments sémantiques par rapport au prédicat. Dans l'exemple, *chien* serait l'agent accomplissant l'action (prédicat), et *dans la rue* serait un autre argument indiquant le lieu où l'action se déroule. Ainsi, imposer des contraintes peut être

pertinent afin de s'assurer que les rôles assignés aux mots respectent certaines règles sémantiques de la langue. Punyakanok *et al.* propose d'imposer une telle structure à l'inférence en utilisant la programmation linéaire en nombres entiers (ILP) [25]. Les contraintes imposées consistent, entre autres, à empêcher la superposition des arguments pour un mot ou que certains rôles sémantiques n'apparaissent qu'une seule fois dans une phrase. Par la suite, Täckström *et al.* propose une méthode utilisant la programmation dynamique [26]. Leur méthode permet d'imposer la plupart des mêmes contraintes qu'avec l'ILP tout en trouvant les mêmes solutions optimales, mais avec une efficacité quatre fois plus grande. Leur méthode de programmation dynamique est formulée en tant qu'un graphe pondéré où le chemin le plus court correspond à la solution optimale trouvée par un ILP. La méthode permet aussi aux contraintes d'être prises en compte lors de l'entraînement. En effet, en utilisant l'algorithme de *forward-backward*, des probabilités marginales peuvent être calculées par rapport au respect des contraintes influençant ainsi les scores de confiance. De ce fait, en appliquant la méthode à l'entraînement et à l'inférence, de meilleurs résultats sont obtenus qu'en appliquant seulement à l'inférence. Une des limitations de cette approche est qu'elle est très spécifique à la tâche du SRL. De plus, peu de contraintes peuvent être imposées. En effet, toutes les contraintes n'ont pas pu être incluses dans leur formulation de programmation dynamique à cause de la complexité combinatoire. Par conséquent, les contraintes restantes ont dû être gérées d'une autre façon, soit en cherchant parmi les k meilleures solutions obtenues de la programmation dynamique.

Lee *et al.* [8] impose des contraintes en utilisant le *gradient-based inference*. Cette méthode consiste à modifier les poids du réseau de neurones lors de la phase d'inférence en utilisant la descente du gradient. Autrement dit, le modèle "continue d'être entraîné" pour respecter les contraintes. La modification des poids est propre à chaque exemplaire. Les contraintes sont imposées sous la forme d'un terme de régularisation dans la fonction de perte représentant le non-respect des contraintes. Plus précisément, les poids W d'un modèle sont d'abord entraînés sans tenir compte des contraintes. Ensuite, pour l'inférence de chaque échantillon individuel, de nouveaux poids W_λ sont initialisés par W. Des itérations sont ensuite effectuées. Une itération consiste premièrement à générer une séquence selon les poids W_λ. Par la suite, la fonction de perte utilisant le terme de régularisation sur les contraintes est calculée. La descente du gradient stochastique est faites afin de mettre à jour W_λ. Les itérations se répètent jusqu'à ce que la séquence générée respecte les contraintes ou lorsque le nombre d'itérations maximum M est atteint. La fonction de perte pénalise aussi selon la différence entre W_λ et W. Ainsi, W_λ est encouragé à rester le plus près possible des poids originaux W. La méthode est testée dans le domaine du NLP dans trois tâches : le SRL, l'analyse de structure syntaxique et la transduction de séquence. Les résultats montrent que le *gradient-based inference* permet de

respecter la grande majorité des contraintes tout en améliorant les performances de précisions pour l'ensemble des tâches. Une limitation de la méthode est qu'elle ne semble pas garantir la satisfaction. En effet, plutôt, la méthode encourage fortement la génération de séquences respectant les contraintes. De plus, la descente du gradient peut être une opération coûteuse dépendamment du nombre d'itérations.

Deutsch *et al.* [6] propose un algorithme général pour ajouter des contraintes à des séquences lors de la phase d'inférence. Les contraintes sont exprimées sous la forme d'automates qui filtreront les valeurs jugées impossibles par ceux-ci. Ce filtrage est fait en fixant la probabilité des valeurs filtrées à zéro. Ainsi, le modèle est assuré de produire une séquence respectant les contraintes. Pour continuer, faire l'intersection des automates des différentes contraintes peut devenir très coûteux et ne pas être nécessaire. En effet, il est possible qu'une contrainte soit satisfaite sans être imposée directement. De ce fait, chaque contrainte est exprimée individuellement par un automate. Pour imposer plusieurs contraintes à la fois, une méthode *active set* est utilisée. La méthode consiste à faire des itérations consistant à générer une séquence jusqu'à ce qu'elle respecte toutes les contraintes. À chaque itération, seulement un sous-ensemble W des contraintes est explicitement imposé. Ce sous-ensemble est initialement vide. De plus, à chaque itération, une seule des contraintes violées par la séquence générée à cette itération est ajoutée au sous-ensemble W. En d'autres mots, l'intersection entre les automates des contraintes dans W est faite. La procédure s'arrête lorsque la séquence générée respecte toutes les contraintes. L'idée est de générer une séquence valide sans nécessairement faire l'intersection de tous les automates des contraintes. Dans le pire cas, toutes les contraintes se trouveront dans le sous-ensemble W. Cependant, les auteurs démontrent que ce cas arrive rarement. Les expérimentations de la méthode sont faites dans le domaine du NLP dans deux tâches : le SRL et l'analyse de structure syntaxique. Les auteurs démontrent qu'il est possible de représenter la plupart des contraintes utilisées en NLP par le biais d'automates. Pour la tâche du SRL, leur méthode corrige les erreurs communes tout en atteignant un score F1 plus élevé de 1,5 point comparativement à un modèle sans contraintes. Pour la tâche d'analyse de structure syntaxique, leur méthode permet de toujours générer une sortie valide aux dépens d'une petite diminution en performance. Enfin, leur algorithme obtient de meilleurs résultats lorsque la grosseur du jeu de données est limitée. Comparativement à notre méthode, comme nous le verrons au prochain chapitre, la leur ne fait que filtrer les valeurs jugées impossibles par les automates dans la distribution de probabilité. Cependant, notre méthode permet en plus de modifier la probabilité marginale des valeurs restantes afin de pousser les valeurs les plus susceptibles de mener à des solutions permises.

CHAPITRE 4 AJOUT DE STRUCTURE LORS DE L'INFÉRENCE

Dans ce chapitre, nous expliquons notre méthode afin d'ajouter une structure à un modèle génératif de séquences lors de la phase d'inférence. La structure est exprimée sous la forme d'un modèle CP. La distribution de probabilité provenant du modèle génératif est combinée avec celle du modèle CP pour obtenir une nouvelle distribution avec laquelle le prochain jeton est échantillonné. Comme modèle génératif, nous nous basons sur le *Chord-conditioned Melody Transformer*, un modèle récent représentatif de l'état de l'art dans le domaine de la génération de musique. Nous nous basons aussi sur MiniCPBP, un solveur CP capable de produire des probabilités marginales sur les valeurs des domaines à travers le processus de *belief propagation*.

4.1 *Chord-conditioned Melody Transformer*

Le *Chord-conditioned Melody Transformer* (CMT) [1] est un modèle permettant de générer une mélodie selon une suite d'accords donnée. Une des particularités du CMT est de générer le rythme et les hauteurs de note de la mélodie séparément. Plus précisément, soit T le nombre total de pas de temps, le modèle génère une séquence de rythme (durées) $r_{1:T} = \{r_1, \ldots, r_T\}$ et une séquence de hauteurs de note $p_{1:T} = \{p_1, \ldots, p_T\}$ de façon autorégressive selon une suite d'accords donnée $c_{1:T} = \{c_1, \ldots, c_T\}$. Il est important de noter que la séquence de rythme est d'abord entièrement générée avant la séquence de hauteurs de note.

Pour la représentation de ces séquences, un vecteur binaire d'une dimension de 12 est utilisé pour représenter un accord. Chaque dimension correspond à une classe de hauteur de note. Ainsi, lorsqu'une classe de hauteur de note est présente dans l'accord, sa dimension correspondante dans le vecteur est mise à 1. Pour le rythme d'une mélodie, un encodage *one-hot* avec une dimension de 3 est utilisé. Les trois valeurs rythmiques correspondent à "début", "maintien" et "silence". Pour les hauteurs de note, un encodage *one-hot* avec une dimension de 50 est utilisé. Les 48 premières dimensions représentent une hauteur de note de MIDI (do^2 à si^5). Les deux dernières dimensions correspondent à "maintien" et "silence". Notez qu'il n'y a pas de représentation "maintien" et "silence" pour les accords. L'unité d'un pas de temps est une double croche.

Comme illustré à la figure 4.1, l'architecture du modèle est composée de trois modules principaux : le *chord encoder* (CE), le RD et le *pitch decoder* (PD). Tout d'abord, le CE est un BLSTM qui encode la suite d'accords $c_{1:T}$ pour obtenir $\tilde{c}_{1:T}$. Ensuite, le RD est une pile de N

blocs d'auto-attention. Lors du pas de temps t, le RD prend en entrée les précédents jetons de rythme générés $r_{1:t-1}$ et la suite d'accords encodée au temps précédant le temps t, $\tilde{c}_{1:t-1}$. Le RD est suivi d'une couche entièrement connectée et d'une couche softmax dans le but d'obtenir une distribution de probabilité sur les différentes valeurs de jeton de rythme. La fonction softmax σ_{SM} transforme un vecteur de nombres $\mathbf{z}$ en probabilités proportionnellement aux valeurs des nombres. La probabilité $\mathbf{z_k}$ d'une des valeurs est obtenue par la formule suivante :

$$\sigma_{SM}(\mathbf{z_k}, \mathbf{z}) = \frac{e^{\mathbf{z_k}}}{\sum_j e^{\mathbf{z_j}}} \tag{4.1}$$

Le PD est aussi une pile de N blocs d'auto-attention. Comme il a été mentionné précédemment, à ce stade, la séquence de rythme est entièrement générée. Ainsi, le RD est réutilisée une dernière fois en tant qu'encodeur pour encoder la séquence de rythme déjà générée. De ce fait, à partir de $r_{1:T}$ et $\tilde{c}_{1:T}$, la séquence de rythme encodée $\tilde{r}_{1:T}$ est obtenue. Par conséquent, lors du pas de temps t, le PD prend en entrée les précédents jetons de hauteur de note générés $p_{1:t-1}$, la séquence de rythme encodée $\tilde{r}_{1:t-1}$ et la suite d'accords encodée au temps précédant le temps t, $\tilde{c}_{1:t-1}$. Tout comme avec le RD, le PD est aussi suivi d'une couche entièrement connectée et d'une couche softmax pour obtenir une distribution de probabilité sur les différentes valeurs de jeton de hauteur de note.

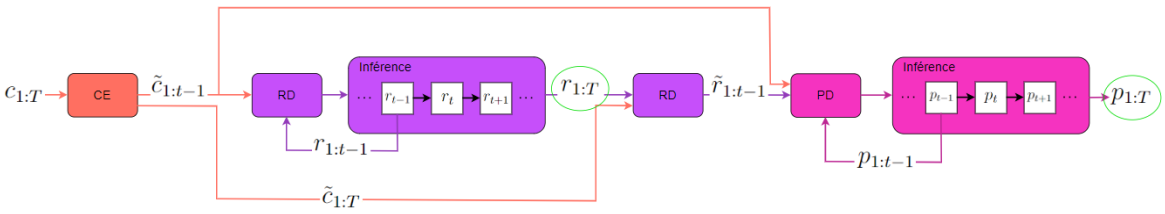

Figure 4.1 Architecture du CMT lors de la génération. Une fois la séquence de rythme générée ($r_{1:T}$), le *rhythm decoder* (RD) est réutilisé en tant qu'encodeur pour aider à générer la séquence de hauteurs de note ($p_{1:T}$).

Entraîner le CMT requiert deux phases. Lors de la première phase, seulement le RD est entraîné. Pour cette phase, les mélodies et les suites d'accords du jeu de données sont transposées entre -5 et +6 demi-tons. On obtient ainsi un jeu de données 12 fois plus gros. Entraîner le RD sur plusieurs gammes rend le modèle plus robuste à différentes classes de hauteur de note. De plus, les vraies étiquettes de rythme des mélodies restent les mêmes, peu importe la transposition. Durant la première phase, seulement la fonction de perte du RD est minimisée. Lors de la deuxième phase, le PD est entraîné à partir du RD préentraîné. Cette fois-ci, les mélodies et les suites d'accords du jeu de données sont transposées dans une seule

gamme pour des raisons d'efficacité de calcul. Par conséquent, le PD est entraîné pour générer des mélodies dans une seule gamme. Par contre, il est toujours possible d'obtenir une autre gamme en transposant la mélodie générée par le bon nombre de demi-tons. Durant cette phase, la fonction de perte minimisée correspond à la somme de la fonction de perte pour le RD et le PD. Par le fait même, les paramètres du RD sont aussi ajustés lors de la deuxième phase.

Lors de la phase de génération, la séquence de rythme est d'abord complètement générée avant celle des hauteurs de note. Chaque séquence est générée de façon autorégressive, jeton par jeton. Plus précisément, à chaque pas de temps, une distribution de probabilité sur les différentes valeurs possibles de jeton est produite. Par la suite, le prochain jeton est échantillonné à partir de cette distribution. Ce processus continue jusqu'à ce que la séquence soit entièrement générée. Pour la séquence de hauteurs de note spécifiquement, comme il y a 50 valeurs possibles de jeton, le prochain jeton est échantillonné à partir d'une distribution normalisée composée du top 5 des probabilités.

4.2 MiniCP

MiniCP [27] est un solveur CP *open-source* léger implémenté en Java. Il a été conçu à des fins d'enseignement et de recherche. Bien que MiniCP ne possède pas toutes les contraintes ou les algorithmes de filtrage, grâce à une habile ingénierie logicielle, son architecture est simple et minimaliste. Par conséquent, il est relativement facile de comprendre son implémentation et d'y ajouter de nouvelles fonctionnalités.

4.3 MiniCPBP

MiniCPBP [28] est un prototype dans lequel de la *belief propagation* (BP) a été ajoutée au solveur MiniCP. La BP est un algorithme à passage de messages dans le but d'effectuer des inférences sur des modèles graphiques. Elle permet d'obtenir une approximation de la distribution marginale des variables individuelles à partir de leur distribution conjointe. Le processus consiste aux nœuds du graphe d'échanger entre eux et de mettre à jour de façon itérative leur croyance sur les probabilités des différentes valeurs qu'une variable puisse prendre. En général, ce processus peut ne pas converger. Cependant, lorsque le graphe est un arbre, le processus converge aux marginales exactes.

Dans le contexte de la CP, cette dernière peut être considérée comme un graphe de facteurs, soit un graphe biparti contenant des nœuds de variable et de contrainte. Chaque nœud de contrainte est relié aux nœuds de variable impliqués dans la contrainte. Les nœuds de

contrainte et de variable s'échangent des messages correspondant à leur croyance (probabilité) locale qu'une valeur d'une variable fait partie d'une solution réalisable. En effectuant ces échanges de façon itérative, il est possible de converger à des marginales approximatives (ou parfois exactes) pour chaque couple valeur-variable.

Par exemple, considérez le CSP suivant :

$$X = \{x_1, \quad x_2, \quad x_3\} \tag{4.2}$$

$$D = \{d_1 = \{0, 1, 2\}, \quad d_2 = \{1, 2, 3\}, \quad d_3 = \{1, 2, 3\}\} \tag{4.3}$$

$$C = \{\texttt{alldifferent}(x_1, x_2, x_3), \quad x_2 + x_3 = 5, \quad x_1 + x_3 > 2\} \tag{4.4}$$

Ce CSP contient trois solutions (x_1, x_2, x_3) : $(0, 2, 3)$, $(1, 2, 3)$, $(1, 3, 2)$. Il est possible de remarquer que la valeur 1 pour la variable x_1 est présente dans deux solutions parmi les trois. Par conséquent, la probabilité marginale de faire partie d'une solution possible pour cette paire variable-valeur est de 67%. Cette probabilité n'indique pas seulement si la paire variable-valeur fait partie d'une solution ou non, elle permet aussi d'informer à quel point cette paire peut mener à une solution réalisable.

De manière formelle, $\mu_{x \to c}(v)$ correspond au message de la variable x à la contrainte c et $\mu_{c \to x}(v)$ représente le message de la contrainte c à la variable x. Les fonctions de message sont définies comme suit :

$$\mu_{x \to c}(v) = \prod_{c' \in N(x) \setminus \{c\}} \mu_{c' \to x}(v) \qquad \forall v \in D(x) \tag{4.5}$$

$$\mu_{c \to x}(v) = \sum_{\boldsymbol{v} : \boldsymbol{v}[x] = v} f_c(\boldsymbol{v}) \prod_{x' \in N(c) \setminus \{x\}} \mu_{x' \to c}(\boldsymbol{v}[x']) \qquad \forall v \in D(x) \tag{4.6}$$

où $D(x)$ est le domaine de x, $N(x)$ est l'ensemble des contraintes dans lesquelles la variable x apparaît, $N(c)$ est l'ensemble des variables impliquées dans la contrainte c, $\boldsymbol{v}$ est un tuple parmi ceux du produit cartésien des domaines de toutes les variables dans $N(c)$, $\boldsymbol{v}[x]$ correspond à la valeur prise par la variable x dans le tuple $\boldsymbol{v}$ et f_c une fonction retournant 1 si le tuple $\boldsymbol{v}$ satisfait la contrainte c, sinon 0.

La marginale non normalisée pour chaque valeur de la variable x est obtenue comme suit :

$$\theta_x(v) = \prod_{c \in N(x)} \mu_{c \to x}(v) \qquad \forall v \in D(x) \tag{4.7}$$

Les croyances initiales de chaque contrainte correspondent à leurs probabilités marginales locales. En d'autres mots, les marginales initiales sont calculées seulement selon l'ensemble

des solutions propre à chaque contrainte individuelle. Par la suite, les variables reçoivent et multiplient les marginales locales provenant des différentes contraintes dans lesquelles elles sont impliquées. À leur tour, les variables retournent aux contraintes leur distribution résultante. De ce fait, les contraintes recalculent leurs marginales locales, mais cette fois-ci, pondérées par la distribution retournée par les variables. Ces itérations d'échange de messages continuent jusqu'à une condition d'arrêt ou convergence. Plus de détails sur l'implémentation de la BP dans MiniCP peuvent être retrouvés dans [28].

4.4 Intégration de la CP et du CMT

Nous présentons maintenant la contribution de ce mémoire. La CP est intégrée au modèle CMT seulement durant la phase de génération. Elle intervient juste avant qu'un jeton soit échantillonné. Les modèles CP sont utilisés à chaque pas de temps pour modifier la distribution de probabilité provenant de la couche softmax (figure 4.2). La distribution provenant des couches softmax ne tiennent compte que du style de musique appris par CMT. Ainsi, les modèles CP vont produire une nouvelle distribution tenant compte aussi des contraintes imposées.

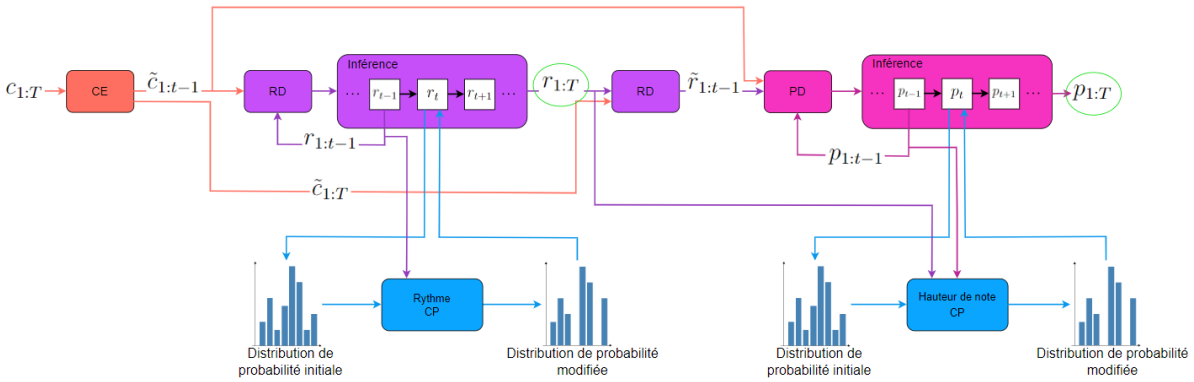

Figure 4.2 Architecture du CMT lors de la génération avec l'ajout des modèles CP. À la génération de chaque jeton, les modèles CP modifient la distribution de probabilité à échantillonner.

Tout d'abord, il est important de noter que la CP n'est utilisée que pour filtrer le domaine du jeton courant à générer par l'entremise des algorithmes des contraintes. Aussi, la CP servira à exécuter des itérations de BP pour obtenir les marginales pour chaque valeur du jeton courant. Par conséquent, la CP n'est pas utilisée pour effectuer un arbre de recherche et trouver une solution réalisable ou optimiser une certaine fonction objectif. La CP est utilisée à chaque pas de temps, et lorsque la séquence de rythme ou de hauteurs de note est entièrement générée, aucune autre opération n'est faite du côté de la CP.

Dans les modèles CP, la séquence de jetons à générer est représentée sous forme d'une séquence de variables à domaine fini. En entrée pour le modèle CP du rythme, les précédents jetons rythmiques générés ainsi que la distribution de probabilité provenant de la couche softmax du CMT pour le jeton courant sont donnés. Les jetons précédents servent à fixer la valeur des variables correspondantes dans le modèle CP. La distribution de probabilité du CMT sert à déclarer une contrainte qui est nécessaire à tous les modèles CP. C'est la contrainte `oracle` :

$$\texttt{oracle}(x_t, D(x_t), p) \tag{4.8}$$

avec x_t la variable correspondant au jeton courant à générer, $D(x_t)$ son domaine et p une fonction de masse fixe sur $D(x_t)$. Dans notre cas, cette fonction de masse p correspond à la distribution de probabilité du CMT donnée en entrée. Pour continuer, la contrainte `oracle` n'impose aucune relation entre les variables. C'est d'ailleurs une contrainte unaire. Elle sert uniquement à contribuer aux passages des messages pendant la BP pour la variable x_t comme toute autre contrainte du modèle CP. Le jeton courant est généré en échantillonnant la distribution correspondant aux marginales résultant de la BP. Par conséquent, c'est par cette contrainte `oracle` que se fait l'intégration du CMT avec la CP. En effet, de cette manière, il est possible de propager la distribution du CMT dans le processus de la BP. Un modèle CP sans la contrainte `oracle` donnerait des marginales ne tenant compte que du respect des contraintes et pas du style de musique appris par le CMT.

Tout comme avec le rythme, pour le modèle CP des hauteurs de note, il faut aussi donner en entrée les précédents jetons de hauteur de note et la distribution de probabilité du CMT. Par contre, il faut aussi donner la séquence de jetons de rythme générée. En effet, lorsque le jeton de rythme au pas de temps t est "maintien" ou "silence", alors le jeton de hauteur de note correspondant doit automatiquement être aussi fixé à 'maintien' ou 'silence'. Par conséquent, certains des futurs jetons de hauteur de note sont déjà fixés. Cela est nécessaire pour que le modèle CP puisse tenir compte du futur et produire des marginales cohérentes selon les contraintes imposées.

Pour continuer, une particularité du solveur MiniCPBP est qu'il permet d'ajuster le poids d'une contrainte dans le modèle CP. Le poids d'une contrainte affecte la façon dont ses marginales sont propagées lors de la BP. Notez que le poids n'affecte pas le filtrage ou la dureté de la contrainte. Étant donné un poids w (la valeur par défaut étant 1), chaque marginale $\theta_x(v)$ pour une valeur v dans le domaine de la variable x est mise en exposant du poids et normalisée sur $D(x)$. La nouvelle marginale obtenue est donc : $(\theta_x(v))^w / \sum_{d \in D(x)} (\theta_x(d))^w$. Par conséquent, un poids $w > 1$ accentue les écarts entre les marginales, alors qu'un poids $0 < w < 1$ les diminue.

Par exemple, soit des marginales suivantes pour une variable d'une contrainte donnée : $\{0 :0{,}2\,;\ 1 :0{,}5\,;\ 2 :0{,}3\}$ et un poids $w = 0,5$. Avant que la contrainte propage ces marginales, elles seront mises en exposant du poids w : $\{0 :0{,}447\,;\ 1 :0{,}707\,;\ 2 :0{,}548\}$. Pour obtenir des marginales sommant à 1, elles sont normalisées : $\{0 :0{,}263\,;\ 1 :0{,}415\,;\ 2 :0{,}322\}$. Parce que le poids de la contrainte est entre 0 et 1, les marginales résultantes sont devenues plus rapprochées entre elles comparativement aux marginales initiales. Nous verrons que jouer avec le poids de la contrainte `oracle` spécifiquement s'avère utile lors des expériences.

CHAPITRE 5 EXPÉRIENCES

Ce chapitre montre les différentes expériences réalisées afin d'évaluer notre méthode présentée au chapitre précédent. Principalement, nous cherchons à ajouter une structure aux mélodies générées par CMT sans toutefois trop s'éloigner du style de musique appris par ce dernier. Dans les prochaines sections, nous verrons qu'il n'est pas évident d'équilibrer ces deux enjeux. Cependant, nous arrivons à trouver une technique permettant de trouver un équilibre raisonnable.

5.1 Configuration expérimentale

Le jeu de données, le prétraitement ainsi que les hyperparamètres pour entraîner le CMT sont les mêmes que ceux employés dans l'article l'introduisant. Le code de ce dernier est disponible sur Github [1]. Ainsi, le jeu de données utilisé pour les expériences est le *Enhanced Wikifonia Leadsheet Dataset* (EWLD) [29] conçu à des fins de recherche. Il contient plus de 5 000 partitions musicales sous le format MusicXML (.mxl) incluant des métadonnées sur le compositeur, le genre, les paroles et autres. Le jeu de données contient plusieurs styles de musique différents, mais majoritairement du Jazz, Pop et Rock. Les ratios d'entraînement, de validation et de test sont de 80%, 10% et 10% respectivement.

Pour entraîner le CMT, celui-ci a besoin de données sous le format MIDI (.mid) contenant une piste pour la mélodie et une autre piste pour la suite d'accords. Par conséquent, le jeu de données a dû être converti sous le format MIDI à l'aide de la librairie Python music21 [30]. Cette librairie permet, entre autres, d'analyser, manipuler et générer du contenu musical. Par la suite, le script Python de prétraitement du CMT a été employé. Celui-ci transpose toutes les chansons en *do* majeur ou en *la* mineur avec une signature rythmique en 4/4. De plus, chaque chanson est divisée en sections de 8 mesures en utilisant une fenêtre coulissante de 4 mesures. L'unité du pas de temps est la double croche. Par conséquent, il y a 16 pas de temps par mesure et 128 (16 x 8) pas de temps dans 8 mesures. Le script enlève aussi du jeu de données les exemplaires de 8 mesures contenant un écart entre la hauteur de note maximale et minimale plus grand que 4 octaves, 2 notes consécutives avec un intervalle plus grand qu'une octave et plus de 32 pas de temps en silence. Notez que le CMT génère des mélodies de 8 mesures dans la gamme de *do* majeur ou de *la* mineur, car il est entraîné sur des exemplaires de 8 mesures dans ces tonalités. Après le filtrage du script Python de prétraitement, le jeu de données est composé d'environ 23 800 exemplaires de 8 mesures pour l'entraînement et 3 150

1. `https://github.com/ckycky3/CMT-pytorch`

chacun pour la validation et le test. De plus, les générations des mélodies sont amorcées par les 4 premiers vrais jetons du jeu de données de test, ce qui correspond au premier temps de musique pour la toute première mesure.

La fonction de perte du RD (L_r) est le *negative log likelihood* (NLL). La fonction de perte du PD (L_p) est le *focal loss* [31]. Cette dernière a été choisie pour compenser le déséquilibre des valeurs des jetons. En effet, dans une séquence de hauteurs de note, il y a davantage de jetons “maintien” et “silence” que de jetons correspondant à une hauteur de note. Ces deux fonctions de perte sont définies comme suit :

$$L_r = -\sum_{t=2}^{T} \log(pr(r_t)) \tag{5.1}$$

$$L_p = -\sum_{t=2}^{T} (1 - pr(p_t))^{\gamma} \log(pr(p_t)) \tag{5.2}$$

où $pr(r_t)$ et $pr(p_t)$ représentent la probabilité de prédiction pour la vraie étiquette pour le RD et le PD respectivement au temps t. Le paramètre γ est mis à 2 et T est le nombre de pas de temps total.

Le modèle CMT est implémenté en utilisant la librairie Pytorch. La dimension des plongements pour les suites d’accords, les rythmes et les hauteurs de note sont 128, 32 et 256 respectivement. La couche de BLSTM a une dimension cachée de 56. Le RD et le PD ont chacun 8 blocs d’auto-attention avec 16 têtes d’attention et une probabilité d’abandon de 20%. Les dimensions cachées du RD et du PD sont 144 et 512 respectivement. Adam est l’optimiseur utilisé avec un taux d’apprentissage de 10^{-4}. Si la fonction de perte sur le jeu de données de validation n’a pas été réduite dans les quatre dernières *epoch*, alors le taux d’apprentissage décroît avec un facteur de 0,5 avec une valeur minimale de 10^{-6}. Tous les *random seeds* sont mis à la valeur de 42.

En ce qui concerne MiniCPBP, le nombre d’itérations de BP est laissé à la valeur par défaut, soit 5. Le mode de propagation est mis à SBP, c’est-à-dire que la propagation de supports (le filtrage) est faite avant la BP. Aucun amortissement a été appliqué sur les messages des variables vers les contraintes. Le code de MiniCPBP est publiquement disponible sur Github [2].

2. `https://github.com/PesantGilles/MiniCPBP`

5.2 Métriques

La génération de musique automatique est un domaine où il est difficile d'évaluer la musique provenant des modèles de manière quantitative. En effet, la musique est très subjective. C'est pourquoi un test d'écoute sur des humains est la méthode préférable. En ce qui concerne les métriques quantitatives utilisées lors des expériences, elles servent à déterminer à quel point les mélodies générées se rapprochent des mélodies du jeu de données de test. Ces métriques sont grandement inspirées dans l'article de CMT [1]. Elles seront expliquées dans les prochaines sous-sections. Notez qu'il n'y a pas de métrique mesurant le respect des contraintes imposées, car elles sont assurées par les modèles CP. En effet, chaque séquence de mélodie générée doit respecter les modèles CP. En pratique, avec un modèle CP complexe, il est possible qu'une séquence partiellement générée ne puisse pas aboutir à une séquence respectant les contraintes. Cependant, ce cas n'est jamais arrivé lors de nos expériences.

5.2.1 Motifs rythmiques

Cette métrique est divisée en deux. Comme première métrique, on y retrouve le nombre de motifs rythmiques d'une mesure différents dans le jeu de données généré et de test. Ensuite, avec l'union des différents motifs rythmiques d'une mesure retrouvés dans le jeu de données généré et de test, la fréquence de chacun de ces motifs est calculée indépendamment pour le jeu de données généré et de test. Deux distributions sont ainsi obtenues. Par la suite, la divergence Jensen-Shannon (JSD) est calculée entre celles-ci [32]. La JSD est une métrique basée sur la divergence de Kullback-Leibler (KLD). Cependant, elle est symétrique et entre 0 et 1. Une valeur de 0 signifie que les deux distributions sont identiques, alors qu'une valeur de 1 signifie qu'elles sont complètement différentes. Ainsi, une valeur la plus proche de 0 est mieux.

5.2.2 *Chord tone ratio*

Le *chord tone ratio* est utilisé pour les hauteurs de note. L'intuition derrière cette métrique est qu'une mélodie harmonieuse devrait contenir des notes se trouvant dans la suite d'accords. Par conséquent, cette métrique consiste à calculer le rapport de notes de la mélodie se trouvant dans les notes de l'accord joué au même moment. Autrement dit, elle calcule à quel point la mélodie est en harmonie avec la suite d'accords. La métrique est aussi calculée seulement pour les notes jouées sur le premier temps d'une mesure. En effet, l'harmonie devrait être plus importante sur ces temps. Plus formellement, soit $p_t \in \{0, ..., 49\}$, la valeur de jeton pour la hauteur de note au temps t, c_t l'ensemble des classes de hauteur de note présentent

dans l'accord au temps t (ex., pour l'accord *do* majeur $c_t = \{0, 4, 7\}$), o_t une valeur binaire à 1 ssi p_t correspond à une valeur de hauteur de note (c'est-à-dire pas 48 ni 49, car elles représentent "maintien" et "silence") et e_t une valeur binaire à 1 ssi c_t est vide (aucun accord joué au temps t). Le *chord tone ratio* (ctr) pour un exemplaire de musique est calculé comme suit :

$$\text{ctr} = \frac{\sum_{t=1}^{T} o_t(1 - e_t) \begin{cases} 1, & \text{si } (p_t \bmod 12) \in c_t \\ 0, & \text{sinon} \end{cases}}{\sum_{t=1}^{T} o_t(1 - e_t)} \tag{5.3}$$

Il est possible de remarquer avec l'équation 5.3 que les notes pour lesquelles aucun accord n'est joué en même temps ne sont pas considérées dans le calcul. La métrique présentée dans les tableaux des prochaines sections est la moyenne du *chord tone ratio* à travers tous les exemplaires de musique d'un jeu de données. Par conséquent, une valeur de *chord tone ratio* similaire à celle du jeu de données de test est mieux.

5.2.3 MGEval *framework*

Yang et Lerch [33] ont proposé le MGEval *framework* permettant d'évaluer objectivement la génération de musique symbolique. La motivation est de faciliter la reproductibilité et la comparaison des différents modèles de génération de musique. Ce *framework* est composé de caractéristiques sur le rythme et sur les hauteurs de notes qui seront extraites pour chaque exemplaire de musique d'un jeu de données. Elles seront expliquées plus en détail dans les prochaines lignes. Notez que les acronymes de ces caractéristiques proviennent de l'anglais.

Caractéristiques de hauteurs de note

1. Total de hauteurs de note (PC) : Nombre total de hauteurs de note différentes dans un exemplaire de musique. Le résultat est un scalaire pour chaque exemplaire de musique.
2. Total de hauteurs de note par mesure (PC/bar) : Nombre total de hauteurs de note différentes à chaque mesure dans un exemplaire de musique. Le résultat est un vecteur de dimension b, où b est le nombre de mesures générées pour chaque exemplaire de musique.
3. Ambitus (PR) : Écart entre la hauteur de note utilisée maximale et minimale en demi-tons pour un exemplaire de musique. Le résultat est un scalaire pour chaque exemplaire de musique.
4. Intervalle moyen de hauteur de note (PI) : Moyenne d'intervalle mélodique en demi-tons entre 2 hauteurs de note consécutives. Le résultat est un scalaire pour chaque

exemplaire de musique.

5. Histogramme de hauteurs de note (PCH) : Histogramme de la fréquence de chacune des 12 classes de hauteur de note, peu importe l'octave pour un exemplaire de musique. La fréquence est calculée proportionnellement au nombre total de pas de temps joué. Le résultat est un vecteur de dimension 12 pour chaque exemplaire de musique.
6. Matrice de transition de hauteurs de note (PCTM) : Matrice 12x12 comptant les transitions d'une hauteur de note à une autre hauteur de note pour un exemplaire de musique. Cette caractéristique peut être utile pour déterminer la tonalité ou le genre de musique.

Caractéristiques de rythme

1. Total de nombre de notes (NC) : Nombre total de notes. Le résultat est un scalaire pour chaque exemplaire de musique.
2. Total de nombre de notes par mesure (NC/bar) : Nombre total de notes à chaque mesure. Le résultat est un vecteur de dimension b, où b est le nombre de mesures générées pour chaque exemplaire de musique.
3. Intervalle moyen d'écart de note (IOI) : Temps moyen entre 2 notes consécutives en seconde. Le résultat est un scalaire pour chaque exemplaire de musique.
4. Histogramme de durée de note (NLH) : Histogramme de la fréquence des différentes durées possibles de note. Les durées considérées sont la ronde, blanche, noir, croche, double croche, blanche pointée, noire pointée, croche pointée, double croche pointée, triolet de noire, triolet de blanches et triolet de croches. Par conséquent, le résultat est un vecteur de dimension 12.
5. Matrice de transition de longueur de note (NLTM) : Matrice 12x12 comptant les transitions d'une durée de note à une autre durée de note pour un exemplaire de musique.

Chaque caractéristique est calculée pour chaque exemplaire de musique d'un jeu de données. Ensuite, une validation croisée est faite. Plus précisément, la distance euclidienne pour chaque caractéristique est calculée entre chaque paire possible d'exemplaires. Lorsque les paires sont constituées d'exemplaire faisant partie d'un même jeu de données, on parlera de distances *intra-set*. Lorsque les paires sont constituées d'exemplaires faisant partie de deux jeux de données différents, c'est alors les distances *inter-set*. Après le processus de validation croisée, un histogramme de distances est obtenu pour chaque caractéristique du *framework*. Par la suite, ces histogrammes sont convertis en distributions de probabilité (PDFs). Par conséquent,

pour chaque caractéristique, un PDF *intra-set* pour le jeu de données de test et un PDF *inter-set* entre le jeu de données de test et le jeu de données générés sont calculés. Pour comparer les exemplaires de musique du jeu de données de test à ceux du jeu de données généré, la KLD et le *overlapping area* (OA) sont calculés entre ces deux PDFs. De ce fait, un faible KLD et un OA élevé indiqueront que le jeu de données généré est similaire à celui du test. Pour une raison computationnelle, seulement 500 exemplaires choisis au hasard pour chaque jeu de données de musique ont été considérés pour la validation croisée.

5.3 Reproduire CMT

Notre première expérience est de reproduire les résultats présentés dans l'article du CMT [1]. Nos résultats sont montrés aux tableaux 5.1 et 5.2. Parmi les métriques dans le tableau 5.1, on retrouve la précision à prédire le prochain jeton lors de l'entraînement. Pour la séquence de hauteurs de note, seulement les jetons où le jeton de rythme correspondant est "début" sont considérés.

Tableau 5.1 Comparaison de nos résultats d'entraînement du CMT à ceux de l'original présentés dans l'article [1].

	CMT (notre)	CMT (article)
Fonction de perte (validation)		
rythme (NLL)	**0,216**	0,256
hauteur note (Focal loss)	0,250	**0,238**
totale	**0,465**	0,494
Précision (validation)		
rythme	**0,913**	0,896
hauteur note	0,444	**0,472**
Chord tone ratio (général)		
test	0,719	0,714
généré	**0,711**	0,725
Chord tone ratio (1er temps)		
test	0,791	0,796
généré	0,777	**0,806**
Motifs rythmiques		
nb motifs rythmiques diff mesure (test)	1424	1219
nb motifs rythmiques diff mesure (généré)	**2127**	2259
divergence Jensen-Shannon	0,1143	**0,0814**

En général, nos résultats sont similaires à ceux présentés dans l'article sauf pour les KLDs du MGEval *framework* où nos résultats sont plus élevés. Pour arriver à ses résultats, il a fallu corriger une erreur dans leur code fourni. En effet, dans le fichier preprocess.py à la ligne 152, il y a une indexation inutile [1 :] dans la structure de données contenant les notes présentes dans une suite d'accords. Cette indexation fait en sorte que le deuxième élément jusqu'au dernier élément inclusivement sont retournés. Par conséquent, il manquera toujours une note dans chaque accord. Malgré tout, dans l'ensemble, nos résultats sont assez similaires pour expérimenter en ajoutant des contraintes à la phase de génération.

Tableau 5.2 Comparaison de nos résultats du CMT à l'original (MGEval *framework*). Les caractéristiques par mesure ne sont pas considérées dans l'article du CMT [1].

	CMT (notre)		CMT (article)	
	KLD	OA	KLD	OA
PC	$1{,}24\times10^{-0}$	0.875	$9{,}75\times10^{-3}$	0,881
PR	$4{,}37\times10^{-1}$	0.916	$5{,}64\times10^{-3}$	0,900
PI	$5{,}80\times10^{-2}$	0.932	$7{,}16\times10^{-3}$	0,945
PCH	$5{,}97\times10^{-1}$	0.972	$1{,}87\times10^{-3}$	0,983
PCTM	$2{,}86\times10^{-1}$	0.873	$8{,}95\times10^{-3}$	0,968
NC	$2{,}41\times10^{-2}$	0.949	$7{,}42\times10^{-3}$	0,973
IOI	$2{,}31\times10^{-1}$	0.971	$3{,}95\times10^{-2}$	0.979
NLH	$9{,}56\times10^{-3}$	0.986	$9{,}87\times10^{-4}$	0,984
NLTM	$3{,}01\times10^{-2}$	0.971	$1{,}78\times10^{-2}$	0,984

5.4 Calibration

Comme il a été expliqué à la section 4.4, les probabilités de prédiction du CMT interagissent avec d'autres probabilités provenant du modèle CP. Ainsi, il peut être important pour les probabilités du CMT d'être calibrées. La calibration est une méthode permettant de transformer les probabilités de prédiction pour qu'elles représentent de vraies probabilités [34]. Les réseaux de neurones tendent à surestimer la prédiction de probabilité comparativement à la vraisemblance des données observées.

Dans le cas de la classification multi-classes, un classificateur calcule un vecteur $\mathbf{z}$ de logits (ou de scores) où $\mathbf{z_k}$ représentent le logit de la classe $\mathbf{k}$. Les probabilités sont ensuite calculées en appliquant une fonction softmax σ_{SM} sur le vecteur de logits $\mathbf{z}$ (équation 4.1).

La prédiction de classe $\hat{y}_i$ et le score de confiance $\hat{p}_i$ sont obtenus comme suit :

$$\hat{y}_i = \text{argmax}_k \left(\mathbf{z_k}\right), \quad \hat{p}_i = \max_k \sigma_{SM}(\mathbf{z_k}, \mathbf{z}) \tag{5.4}$$

Platt scaling est une méthode permettant de calibrer les scores de confiance d'un classificateur préentraîner [35]. Nous avons testé deux variations du *Platt scaling* multi-classes en nous basant sur les meilleures performances de calibration dans [34] et [36]. La première est le *matrix scaling* qui consiste à appliquer une transformation linéaire des logits d'un classificateur préentraîné à l'aide d'une matrice de poids $\mathbf{W}$ et un vecteur de biais $\mathbf{b}$. Cette transformation sur les logits s'applique juste avant la fonction softmax. Ainsi, possiblement la nouvelle prédiction de classe $\hat{y}_i^{'}$ et le score de confiance calibré $\hat{q}_i$ sont obtenus comme suit :

$$\hat{y}_i^{'} = \text{argmax}_k \left(\mathbf{W}\mathbf{z_k} + \mathbf{b}\right), \quad \hat{q}_i = \max_k \sigma_{SM}(\mathbf{W_k}\mathbf{z_k} + \mathbf{b_k}, \mathbf{W}\mathbf{z} + \mathbf{b}) \tag{5.5}$$

Les paramètres $\mathbf{W}$ et $\mathbf{b}$ sont optimisés en minimisant la fonction de perte NLL sur le jeu de données de validation. Notez que les paramètres du classificateur préentraîné restent fixes tout au long de la calibration.

La deuxième variation testée du *Platt scaling* est le *Temperature scaling*. Pour cette méthode, seulement un paramètre scalaire $T > 0$ est utilisé comme suit :

$$\hat{y}_i^{'} = \text{argmax}_k \left(\frac{\mathbf{z_k}}{T}\right), \quad \hat{p}_i = \max_k \sigma_{SM}\left(\frac{\mathbf{z_k}}{T}, \frac{\mathbf{z}}{T}\right) \tag{5.6}$$

L'*expected calibration error* (ECE) [37] est une métrique permettant d'évaluer la calibration d'un modèle. Cette métrique divise d'abord les prédictions en M intervalles égaux. Chaque intervalle I_m rassemble les prédictions dont le score de confiance $\hat{p}_i$ fait partie de l'intervalle $(\frac{m-1}{M}, \frac{m}{M}]$. Définissons B_m, l'ensemble des indices des exemplaires dans l'intervalle I_m, $\text{acc}(B_m)$, la précision des prédictions dans B_m et $\text{conf}(B_m)$ le score de confiance moyen dans B_m :

$$\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} 1(\hat{y}_i = y_i) \tag{5.7}$$

$$\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i \tag{5.8}$$

Ainsi, l'ECE est la moyenne pondérée de la différence entre la précision et le score de confiance

moyen de chaque intervalle :

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \Big| \text{acc}(B_m) - \text{conf}(B_m) \Big| \tag{5.9}$$

où $\hat{y}_i$ est la prédiction de classe de l'exemplaire i, y_i sa vraie classe et n le nombre d'exemplaires total. Une valeur plus petite de l'ECE indique une meilleure calibration.

Le tableau 5.3 montre les résultats des deux méthodes de calibration sur le jeu de données de test. La calibration est faite sur le RD. Tout d'abord, le modèle CMT par défaut est déjà très bien calibré. Ceci est remarquable par le petit ECE de 0,4%. En général, les deux variations du *Platt scaling* ont toutes les deux été capables de légèrement améliorer la calibration du CMT en diminuant le NLL, l'ECE et en augmentant la précision (*accuracy*). Le *matrix scaling* a donné de meilleurs résultats globalement que le *Temperature scaling*.

Comme le modèle CMT est déjà très bien calibré, et que les méthodes de calibration n'apportaient que de légères améliorations, nous avons continué les expériences avec un modèle CMT non calibré.

Tableau 5.3 NLL, précision et ECE résultant des deux méthodes de calibration effectuées sur le RD.

	Sans Calibration	Matrix Scaling	Temp. Scaling
NLL (validation)	0,4652	**0,4644**	0,4656
Préc. Hauteur note (test)	44,26%	**44,33%**	44,24%
Préc. Rythme (test)	90,89%	90,89%	**90,91%**
ECE (test)	0,4%	**0,1%**	0,2%

5.5 Nombre de notes croissant à chaque mesure

Pour la toute première expérience d'ajouter une structure au CMT, nous avons implémenté une première contrainte par un modèle CP en utilisant MiniCPBP. La contrainte consiste à avoir un nombre de notes dans une mesure strictement supérieur au nombre de notes de la mesure précédente. Par conséquent, elle affecte uniquement le rythme de la mélodie. La contrainte est seulement imposée sur des groupes de quatre mesures. Comme mentionné à la section 5.1, CMT génère des mélodies de huit mesures. De ce fait, lorsque la contrainte est appliquée sur des groupes de quatre mesures, elle sera imposée indépendamment sur les mesures 1,2,3,4 et mesures 5,6,7,8. Autrement dit, la contrainte se "réinitialise" après quatre mesures. Cette contrainte a été choisie, car elle affecte la génération globalement. La

contrainte n'est peut-être pas la plus réaliste musicalement, mais toutefois raisonnable. Afin d'avoir une simple première idée sur les performances de notre méthode, nous n'étudions pas l'effet d'étendre la contrainte sur plus de mesures consécutives, et nous n'utilisons pas les métriques du MGEval *framework*.

De façon formelle, nous définissons $CSP_{ryhtme1}(X \cup O, D, C)$ avec $X = \{x_1, \ldots, x_{128}\}$ l'ensemble de variables représentant chacun des 128 jetons de la séquence de rythme. $D(x_i) = \{0,\ 1,\ 2\}$ pour chaque $x_i \in X$, car il y a trois valeurs possibles pour les jetons de rythme comme mentionné à la section 4.1. Les variables $O = \{o_1, \ldots, o_8\}$ avec $D(o_i) = \{0, \ldots,\ 16\}$ représentent le nombre d'occurrences du jeton "début" dans chaque mesure. Il y a 16 pas de temps par mesure. Par conséquent, il peut y avoir jusqu'à 16 jetons "début" dans une mesure.

Les contraintes C sont les suivantes :

$$\textbf{among}(x_{((i-1)*16)+1}, ..., x_{i*16}, \text{“jeton\_début”}, o_i) \qquad \forall i \in \{1, ..., 8\} \tag{5.10}$$

$$o_i < o_{i+1} \qquad \forall i \in \{1, ..., 7\} \quad \text{si } (i \bmod b) \neq 0 \tag{5.11}$$

Chaque contrainte `among` associe le nombre de jetons "début" dans la mesure i à la variable o_i. La contrainte de l'équation 5.11 spécifie que la mesure courante doit avoir un nombre de notes strictement inférieur à la mesure suivante. La valeur b représente le nombre de mesures sur lesquelles la contrainte est appliquée (ici quatre). Ainsi, la condition $(i \bmod b) \neq 0$ permet à la contrainte de se "réinitialiser" après chaque groupe de b mesures.

5.5.1 Multiplication vs contrainte `oracle`

Avant de regarder les mélodies générées, nous avons pensé à une autre méthode plus simple pour combiner les probabilités marginales du CMT à celles du modèle CP qui ne nécessite pas l'usage de la contrainte `oracle`. La méthode consiste à multiplier les marginales provenant du modèle CP (qui ne tiennent compte que du respect des contraintes) aux marginales du CMT élément par élément. La nouvelle distribution est ensuite normalisée, et le prochain jeton est échantillonné. Ainsi, pour cette expérience, ces méthodes ont été comparées. Pour ce faire, les deux méthodes ont été utilisées en même temps lors de la phase de génération en imposant la contrainte d'avoir un nombre croissant de notes. Plus précisément, une première distribution pour le prochain jeton est obtenue avec la méthode de multiplication. Ensuite, une deuxième distribution est obtenue avec la méthode utilisant la contrainte `oracle` non pondérée (poids à 1,0). La KLD est par la suite calculée entre ces deux distributions. Pour progresser dans la génération des séquences, le prochain jeton est arbitrairement échantillonné avec la méthode

de multiplication. Par conséquent, la moyenne de la KLD est calculée à travers toutes les distributions de probabilité calculées pour la génération de tous les jetons de rythme de chacun des 3 150 exemplaires générés. Ainsi, la moyenne de la KLD obtenue est 0,036. Par conséquent, l'utilisation de l'une ou l'autre méthode ne semble pas avoir une grande différence sur la distribution de probabilité sur les jetons de rythme. De ce fait, dans les prochaines expériences, la méthode utilisant l'`oracle` sera utilisée. Notez que ce résultat pourrait différer en faisant la même expérience avec les jetons de hauteur de note. En effet, il y a beaucoup plus de jetons de hauteur de note que de jetons de rythme (50 vs 3 respectivement). Il est aussi possible que le résultat diffère avec un modèle CP plus complexe, soit un modèle contenant davantage de contraintes.

5.5.2 Contrainte `oracle` (non pondérée)

Comme première expérience, les probabilités marginales provenant du CMT sont données au modèle CP à travers la contrainte `oracle`, et des itérations de BP sont faites. Le poids de la contrainte `oracle` est fixé à 1,0, soit la valeur par défaut, tout au long de la génération.

Le tableau 5.4 montre les différentes métriques concernant les motifs rythmiques obtenus en imposant la contrainte. Le nombre de motifs rythmiques différents atteint 5563 comparativement à 1424 dans le jeu de données de test et à 2127 pour une génération de mélodies sans contraintes (tableau 5.1). De plus, la JSD augmente aussi de 0,114 à 0,449. Par conséquent, les mélodies générées sont considérablement différentes rythmiquement du jeu de données de test.

Tableau 5.4 Métriques rythmiques des mélodies générées avec un nombre de notes croissant à chaque mesure en utilisant la valeur par défaut (1,0) du poids de l'`oracle`.

	Générées	Test
Nombre de motifs rythmiques différents d'une mesure	5563	1424
Divergence Jensen-Shannon	0,449	-

Cette différence rythmique peut être expliquée, entre autres, par la nature de la contrainte. En effet, la contrainte imposée est retrouvée dans seulement un exemplaire du jeu de données de test. De plus, la figure 5.1 montre la fréquence des mesures contenant différents nombres de notes dans le jeu de données de test. Il est possible de remarquer que la majorité des mesures contiennent entre une et quatre notes. Par conséquent, si la première mesure contient quatre notes, le modèle se retrouve à devoir générer des mesures contenant plus de quatre notes. De ce fait, les mélodies générées s'éloignent davantage du jeu de données de test.

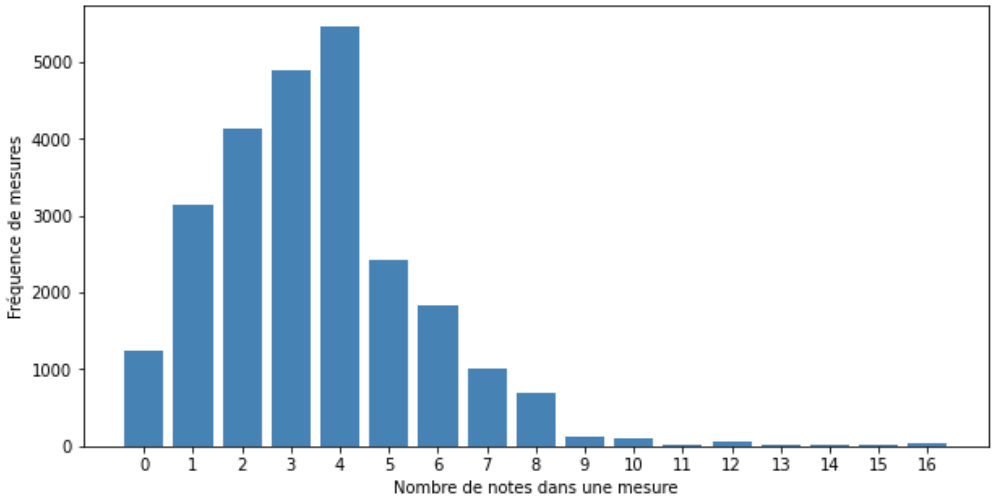

Figure 5.1 Fréquence des mesures contenant différents nombres de notes dans le jeu de données de test.

Problème de procrastination

Un autre résultat est donné à la figure 5.2. Le *heat map* (à droite) montrent combien de fois, parmi le top 10 des motifs rythmiques d'une mesure étant la $i$ème mesure dans le groupe de b mesures, le jeton "début" a été trouvé à un certain pas de temps d'une mesure. La somme des fréquences de ces motifs rythmiques est aussi donnée à la colonne Freq. La figure 5.3 permet de mieux visualiser comment le *heat map* a été produit.

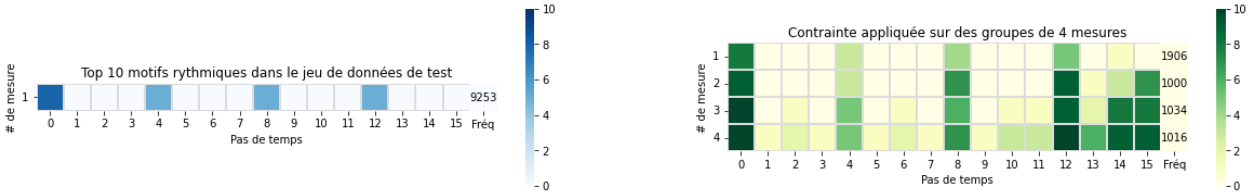

Figure 5.2 *Heat map* des tops 10 des motifs rythmiques avec un nombre de notes croissant à chaque mesure en utilisant la valeur par défaut (1,0) du poids de l'`oracle`.

Dans la figure 5.2, il est possible de remarquer une zone foncée en bas à droite. Par conséquent, cela indique que la plupart des notes sont jouées vers la fin d'une mesure. Ce phénomène est plus important, plus on avance dans les mesures. De ce fait, il semble que le modèle ignore la contrainte lors de la génération jusqu'à la fin où il n'a plus d'autre choix que d'ajouter des notes pour la respecter. En d'autres mots, le respect des contraintes ne presse pas assez lors

de la génération sauf à la fin. Un placement plus uniforme des notes est souhaitable. Nous allons voir que ce problème arrive souvent pour des contraintes à long terme comme celle dans cette expérience. Nous appelons ce problème le *problème de procrastination*. Dans les prochaines sections, nous montrons les différents essais afin de gérer ce problème.

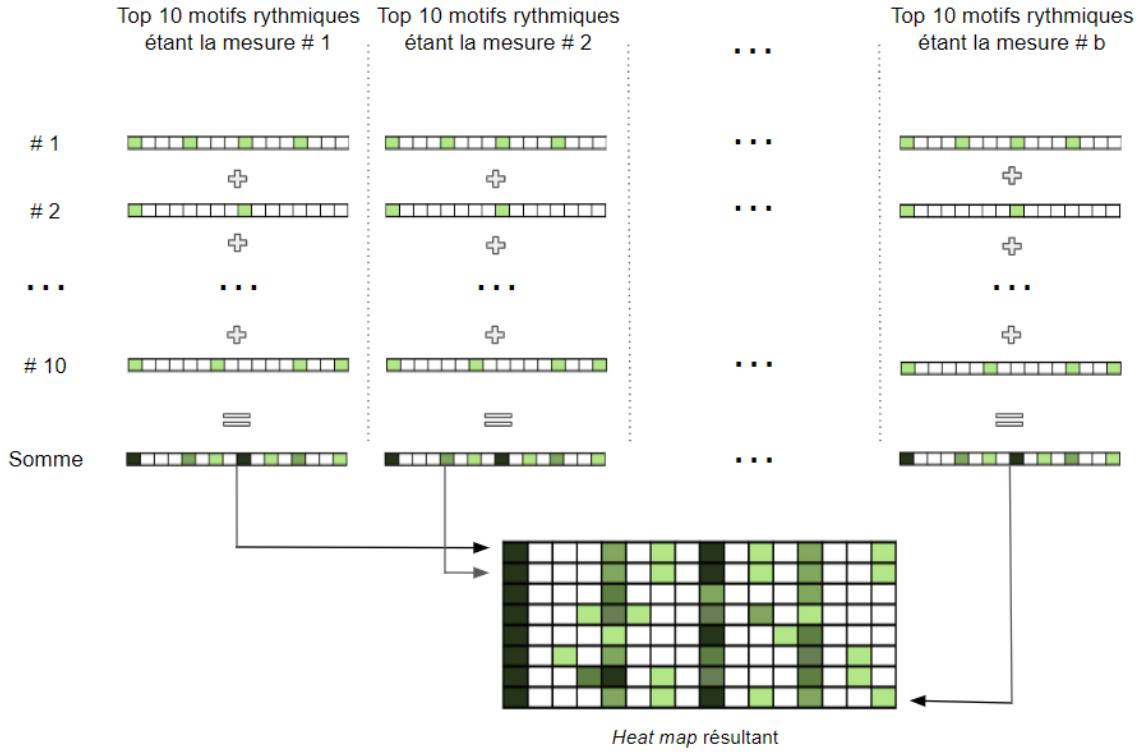

Figure 5.3 Visualisation de la création d'un *heat map* pour le top 10 des motifs rythmiques. Chaque ligne est divisée en 16 pour représenter les 16 jetons dans une mesure. Les teintes de vert représentent l'intensité de la présence d'un jeton "début" à l'index d'une mesure donné.

5.5.3 Diminution fixe du poids de la contrainte `oracle`

Pour cette prochaine expérience, nous essayons de donner plus d'impact au respect des contraintes en donnant moins d'impact aux probabilités marginales provenant du CMT. Ceci peut être accompli en abaissant le poids de la contrainte `oracle` à une valeur fixe. En effet, la contrainte `oracle` correspond aux marginales du CMT qui seront propagées lors du BP. En abaissant son poids, ces marginales auront moins tendance à pousser une certaine valeur de jeton, car elles seront plus rapprochées entre elles (comme vu dans l'exemple à la fin de la section 4.4). De ce fait, les marginales correspondantes au respect des contraintes auront davantage d'influence sur les valeurs générées. Différentes valeurs fixes du poids de `oracle` entre 0,0 et 1,0 ont été essayées.

Tableau 5.5 Métriques rythmiques des mélodies générées avec un nombre de notes croissant à chaque mesure en utilisant différentes valeurs du poids de la contrainte `oracle`.

	Poids de la contrainte `oracle`									Test
	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	
Nombre de motifs rythmiques différents d'une mesure	25163	24916	23925	21974	18924	15107	11681	8972	6953	1424
Divergence Jensen-Shannon	0,994	0,964	0,894	0,795	0,700	0,604	0,529	0,484	0,458	-

Les résultats des *heat maps* des différentes configurations sont montrés à la figure 5.4. Avec un poids entre 0,5 et 1,0, il est toujours possible de qualitativement remarquer le *problème de procrastination*. À partir de la valeur 0,4, le placement des notes commence à devenir beaucoup plus uniforme. Cependant, les fréquences des tops 10 des motifs rythmiques sont très basses (par ex. seulement 20 occurrences parmi toutes les mélodies générées). De plus, au tableau 5.5, on retrouve près de 22 000 différents motifs rythmiques avec une JSD atteignant presque 0,80 pour un poids de 0,4. Par conséquent, les mélodies générées sont encore plus différentes rythmiquement du jeu de données de test. Ceci peut être expliqué par le fait que le poids de la contrainte `oracle` est abaissé. Ainsi, surtout avec un faible poids comme 0,4, la génération se base peu sur le style appris par CMT. Dès lors, il semble difficile de trouver le bon équilibre entre la mitigation du *problème de procrastination* et la similarité avec le jeu de données de test.

5.5.4 Variation du poids de la contrainte `oracle`

Au lieu de garder le poids de l'`oracle` fixe, nous expérimentons à le faire varier lors de la génération. Le poids pourrait augmenter ou diminuer de façon monotone. Une intuition peut se faire pour les deux méthodes. En effet, la génération pourrait commencer avec un faible poids de la contrainte `oracle` pour permettre le respect des contraintes de se faire gérer plus tôt. Ensuite, le poids augmentera afin de hausser la similarité avec le jeu de données de test. D'une autre part, la génération pourrait commencer avec le poids par défaut (1,0). Par la suite, le poids diminue dans le but de compenser la faible pression des contraintes.

Deux méthodes ont été testées pour augmenter (ou diminuer) le poids de l'oracle au cours de la génération. La première méthode est d'augmenter (ou diminuer) le poids à un taux constant après chaque jeton généré. Plus précisément, un poids maximal et minimal est donné. Par la suite, le taux est calculé en divisant la différence de ces deux valeurs par le nombre de jetons à générer. La deuxième méthode consiste à augmenter (ou diminuer) le poids après

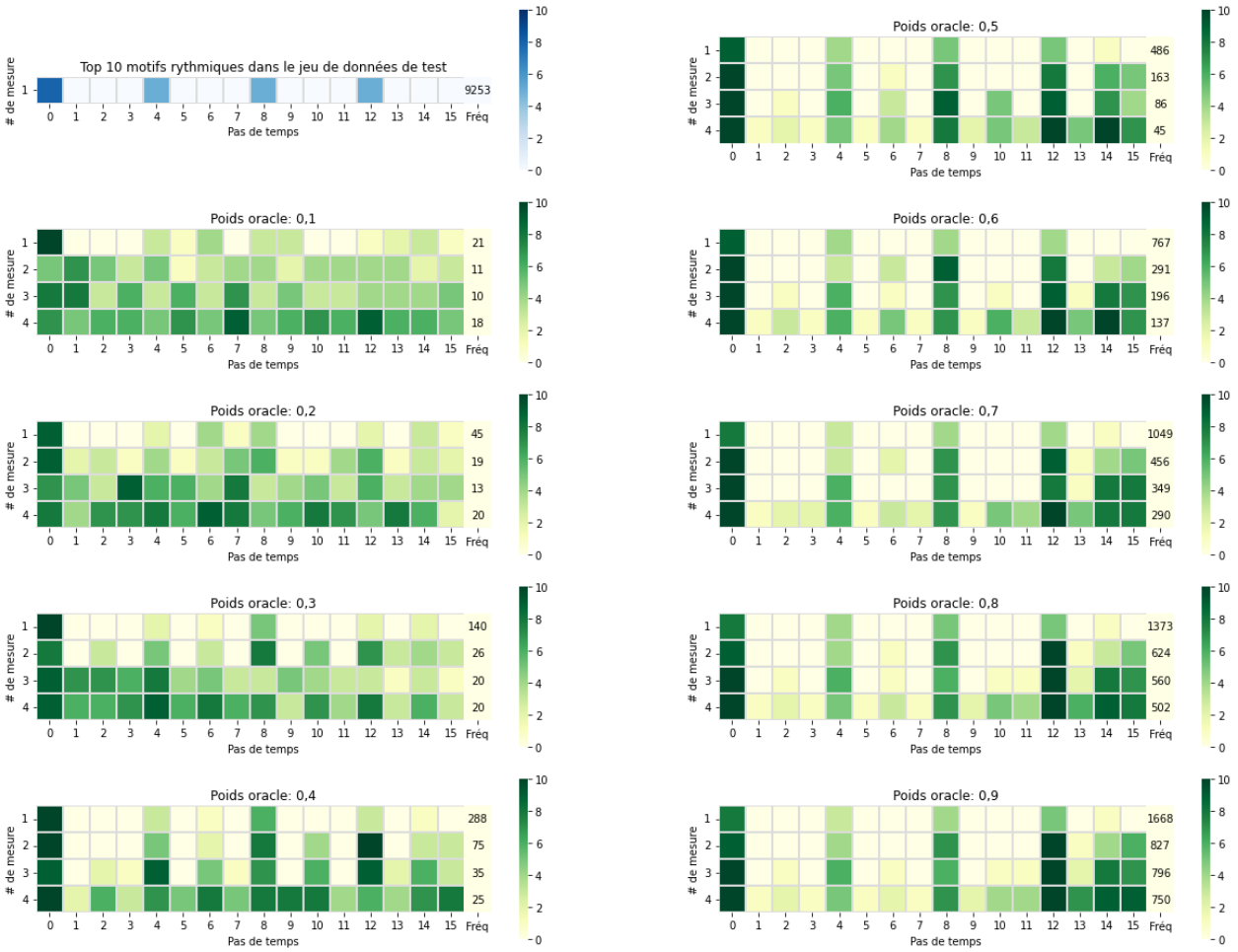

Figure 5.4 *Heat maps* des tops 10 des motifs rythmiques avec un nombre de notes croissant à chaque mesure en utilisant une valeur diminuée fixe du poids de l'`oracle`.

chaque mesure. En d'autres mots, le poids reste fixe pour des jetons d'une même mesure. La valeur du poids à chaque mesure est déterminée manuellement avant la génération. Plus précisément, une liste spécifiant le poids exact pour chacune des quatre mesures est donnée au modèle.

Les différentes expériences et configurations pour chaque méthode sont montrées au tableau 5.6 et à la figure 5.5. Selon ces résultats, le *problème de procrastination* est toujours présent pour les deux méthodes. Cependant, en général, diminuer le poids lors de la génération a donné de meilleurs résultats que de l'augmenter.

Tableau 5.6 Métriques rythmiques des mélodies générées avec un nombre de notes croissant à chaque mesure en utilisant différentes techniques de variation du poids de la contrainte `oracle` lors de la génération. Abbréviation : Augmentation Graduelle (G_up), Augmentation Manuelle (M_up), Diminution Graduelle (G_down), Diminution Manuelle (M_down). Pour la variation graduelle, la valeur maximale et minimale sont données. Pour la variation manuelle après chaque mesure, les valeurs de la liste donnée sont précisées.

	Nb motifs rythmiques diff mesure	Divergence Jensen-Shannon
G_up		
(1) 0,4 à 1,0	14 333	0,637
(2) 0,3 à 1,0	16 999	0,709
(3) 0,3 à 0,8	19 574	0,758
M_up		
(1) 0,4 ; 0,6 ; 0,8 ; 1,0	15 552	0,692
(2) 0,3 ; 0,4 ; 0,6 ; 1,0	20 948	0,829
(3) 0,3 ; 0,4 ; 0,6 ; 0,8	21 506	0,832
G_down		
(1) 1,0 à 0,4	11 913	0,493
(2) 1,0 à 0,3	13 515	0,518
(3) 0,8 à 0,3	16 211	0,587
M_down		
(1) 1,0 ; 0,8 ; 0,6 ; 0,4	12 598	0,491
(2) 1,0 ; 0,6 ; 0,4 ; 0,3	16 311	0,550
(3) 0,8 ; 0,6 ; 0,4 ; 0,3	17 193	0,591

En effet, le nombre de motifs rythmiques différents et la JSD sont plus petits. De plus, la méthode où les poids sont spécifiés manuellement donne une meilleure performance que la méthode avec un taux décroissant constant. En effet, par exemple, pour la configuration M_down (2), le *problème de procrastination* est moins sévère que toutes les autres configura-

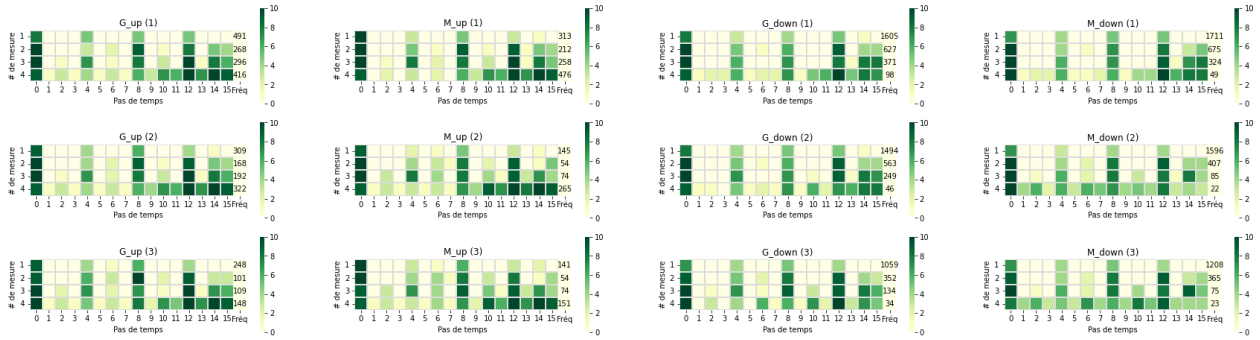

Figure 5.5 *Heat maps* des tops 10 des motifs rythmiques avec un nombre de notes croissant à chaque mesure en utilisant différentes méthodes de varier le poids de l'`oracle` lors de la génération.

tions. Son nombre de motifs rythmiques différents et sa JSD sont toujours élevés (16 311 et 0,550 respectivement). En revanche, ces métriques sont beaucoup moins hautes que d'essayer de résoudre le *problème de procrastination* avec un poids de l'`oracle` réduit et fixe (ex., poids fixe de 0,4). Par conséquent, varier le poids de la contrainte `oracle` a permis de trouver un meilleur équilibre entre la mitigation du *problème de procrastination* et la similarité avec le jeu de données de test.

5.5.5 Décroissance géométrique du poids de la contrainte `oracle` après chaque mesure

Bien que la méthode nécessitant de spécifier le poids manuellement pour chaque mesure donne de meilleurs résultats, elle n'est pas très pratique. En effet, il peut être compliqué de trouver quel poids spécifique utiliser pour chaque mesure. Une particularité parmi les meilleures performances de cette méthode est que le poids décroît rapidement au début. Ceci pourrait expliquer pourquoi une décroissance du poids avec un taux constant arrive à de moins bons résultats. De ce fait, une formule capable de potentiellement répliquer la progression trouvée parmi les meilleurs résultats est la décroissance géométrique. Dans nos expériences, le poids commence à 1, la raison r est entre 0 et 1 et le poids est multiplié par la raison après la fin de génération de chaque mesure.

Les différentes valeurs r essayées ont été déterminées selon la valeur du poids à la deuxième mesure en se basant sur les résultats du M_down de la section précédente. Les résultats des différentes valeurs r sont présentés au tableau 5.7 et à la figure 5.6. Les valeurs avec une meilleure mitigation du *problème de procrastination* sont $r = 0,55$ et $r = 0,60$. En effet, leur *heat map*, vers la fin, contient moins de cases très foncées comparativement à $r = 0,70$ par

Tableau 5.7 Métriques rythmiques des mélodies générées avec un nombre de notes croissant à chaque mesure en utilisant différentes valeurs de raison r afin de diminuer le poids de la contrainte `oracle` de façon géométrique après chaque mesure générée. Plus la valeur r est petite, plus le poids est diminué agressivement.

	Valeurs de raison r					Test
	0,50	0,55	0,60	0,65	0,70	
Nb motifs rythmiques diff mesure	18 767	17 908	16 980	15 791	14 568	1424
Divergence Jensen-Shannon	0,608	0,586	0,565	0,539	0,520	-

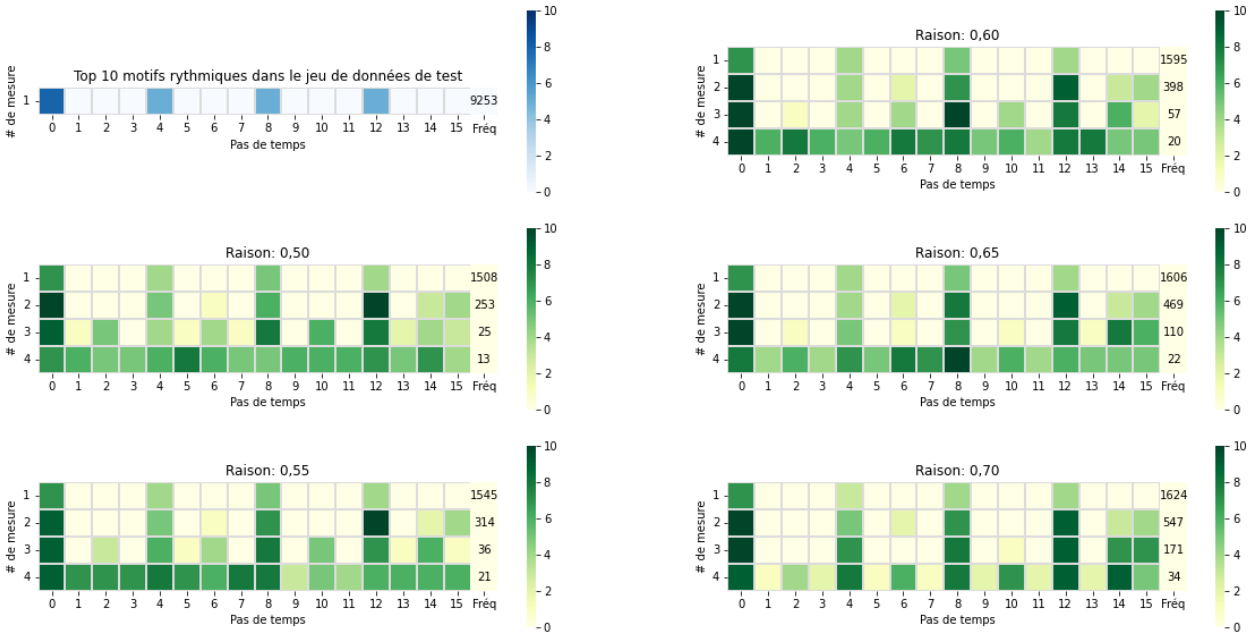

Figure 5.6 *Heat maps* des tops 10 des motifs rythmiques avec un nombre de notes croissant à chaque mesure en utilisant différentes valeurs de raison r afin de diminuer le poids de la contrainte `oracle` de façon géométrique après chaque mesure générée. Plus la valeur r est petite, plus le poids est diminué agressivement.

exemple. Le nombre de motifs rythmiques différents et la JSD obtenus sont 17 908 ; 0,586 et 16 980 ; 0,566 respectivement. Ces résultats sont relativement équivalents à ceux obtenus avec la méthode M_down (2). Ainsi, il est plus pratique d'utiliser cette formule au lieu de spécifier le poids à chaque mesure manuellement. Notez qu'une diminution moins agressive (r plus grand) du poids de l'`oracle` résulte toujours à un nombre de motifs plus petit et à une meilleure JSD. Cependant, il est aussi important de vérifier si le *problème de procrastination* est assez mitigé. C'est pourquoi, malgré les meilleures métriques, une raison $r = 0,70$, par exemple, n'est pas considérée comme un meilleur équilibre. En bref, l'objectif est de trouver la raison r la plus élevée possible qui mitige le problème de procrastination.

5.5.6 Décroissance géométrique du poids de la contrainte `oracle` après chaque jeton

Le succès de la décroissance après chaque mesure peut possiblement être attribué au fait que la contrainte imposée s'applique sur chaque mesure. Ainsi, afin d'avoir une méthode plus générale selon la contrainte choisie ou selon le domaine d'application, un autre type de décroissance géométrique est testé. Il consiste à diminuer le poids après la génération de chaque jeton.

Tableau 5.8 Métriques rythmiques des mélodies générées avec un nombre de notes croissant à chaque mesure en utilisant différentes valeurs de raison r afin de diminuer le poids de la contrainte `oracle` de façon géométrique après chaque jeton généré. Plus la valeur r est petite, plus le poids est diminué agressivement.

	Valeurs de raison r					Test
	0,975	0,981	0,985	0,989	0,991	
Nb motifs rythmiques diff mesure	17 869	15 540	13 511	11 163	9912	1424
Divergence Jensen-Shannon	0,608	0,555	0,518	0,485	0,470	-

Les différentes valeurs r essayées ont été déterminées pour qu'après quatre mesures générées (64 jetons), le poids atteigne environ une certaine valeur (soit 0,2 ; 0,3 ; 0,4 ; 0,5 et 0,6). Les résultats de ces valeurs sont présentés au tableau 5.8 et à la figure 5.7. La valeur ayant eu un meilleur équilibre est $r = 0,975$. Le nombre de motifs rythmiques différents et la JSD atteignent 17 869 et 0,608 respectivement. Ces métriques sont relativement assez proches de la configuration M_down (2) ou de la décroissance géométrique après chaque mesure. Ainsi, la formule géométrique semble convenablement être une formule générale pour mitiger le *problème de procrastination*. De plus, elle ne contient qu'un seul paramètre à ajuster.

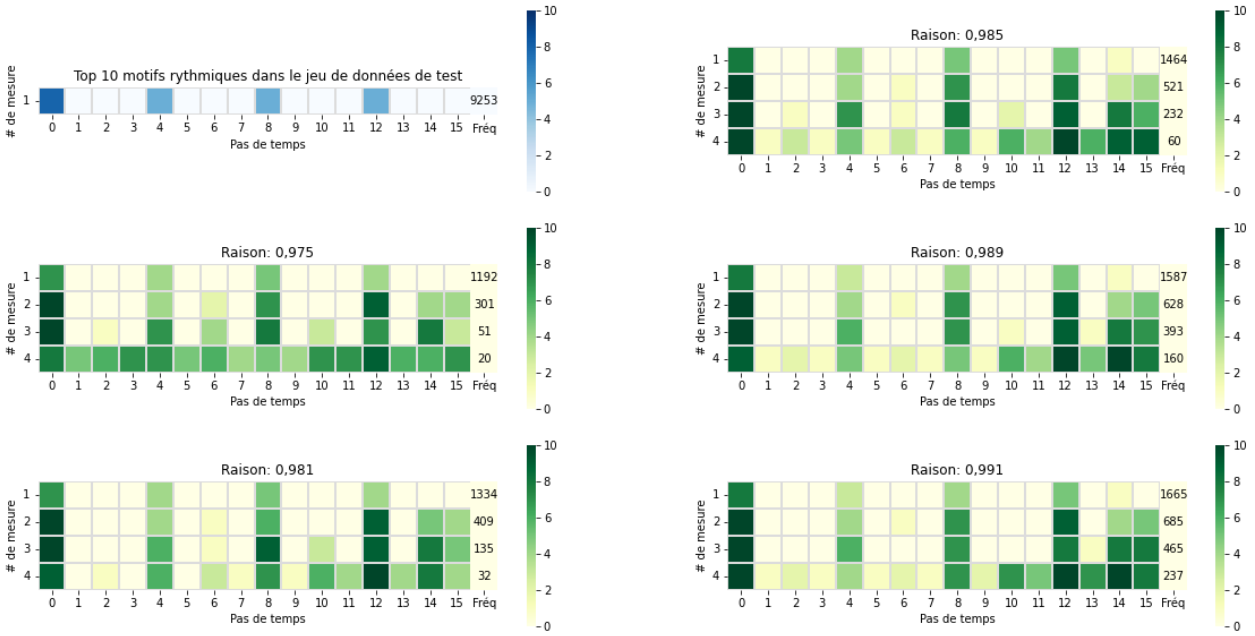

Figure 5.7 *Heat maps* des tops 10 des motifs rythmiques avec un nombre de notes croissant à chaque mesure en utilisant différentes valeurs de raison r afin de diminuer le poids de la contrainte `oracle` de façon géométrique après chaque jeton généré. Plus la valeur r est petite, plus le poids est diminué agressivement.

5.6 Nombre de notes différent par mesure

Nous expérimentons maintenant avec une autre contrainte. La contrainte des expériences précédentes était trop en opposition avec le corpus de test (comme vu à la figure 5.1). Ainsi, il était difficile de juger si l'équilibre entre la mitigation du *problème de procrastination* et de la similarité avec le jeu de données de test était raisonnable ou non. Par conséquent, nous avons implémenté une autre contrainte un peu plus réaliste avec MiniCPBP. La contrainte consiste à avoir un nombre différent de notes à chaque mesure. Tout comme la contrainte des expériences précédentes, elle affecte seulement le rythme de la mélodie et la génération globalement. Pour les prochaines expériences, nous allons aussi étudier l'effet à long terme. En d'autres mots, différentes expériences ont été faites en imposant la contrainte sur des groupes de quatre à huit mesures. Par exemple, lorsque la contrainte est appliquée sur des groupes de cinq mesures, la contrainte sera imposée indépendamment sur le groupe de mesures 1,2,3,4,5 et le groupe 6,7,8. En d'autres mots, les nombres possibles de notes pour une mesure se "réinitialise" après cinq mesures. De plus, nous allons commencer à utiliser les métriques du MGEval *framework* afin d'avoir un ensemble de métriques plus riche et complet. Notez

que cette contrainte reste toujours sujette à s'écarter du corpus selon la figure 5.1. En effet, lorsque la contrainte est imposée sur plus de quatre mesures, même dans le meilleur cas, le modèle doit générer des mesures contenant plus de quatre notes, s'éloignant ainsi du corpus de test.

De façon formelle, nous définissons $CSP_{ryhtme2}(X \cup O, D, C)$ avec $X = \{x_1, \ldots, x_{128}\}$ l'ensemble de variables représentant chacun des 128 jetons de la séquence de rythme. $D(x_i) = \{0, 1, 2\}$ pour chaque $x_i \in X$, car il y a trois valeurs possibles pour les jetons de rythme comme mentionné à la section 4.1. Les variables $O = \{o_1, \ldots, o_8\}$ avec $D(o_i) = \{0,\ldots, 16\}$ représentent le nombre d'occurrences du jeton "début" dans chaque mesure. Il y a 16 pas de temps par mesure. Par conséquent, il peut y avoir jusqu'à 16 jetons "début" dans une mesure.

Les contraintes C sont les suivantes :

$$\texttt{among}(\{x_{((i-1)\cdot 16)+1}, \ldots, x_{i\cdot 16}\}, \text{“jeton\_début”}, o_i) \quad \forall i \in \{1, \ldots, 8\} \tag{5.12}$$

$$\texttt{alldifferent}(o_{((i-1)\cdot b)+1}, \ldots, o_{\min(i\cdot b, 8)}) \quad \forall i \in \left\{1, \ldots, \left\lceil \frac{8}{b} \right\rceil \right\} \tag{5.13}$$

$$\texttt{oracle}(x_t, D(x_t), p) \tag{5.14}$$

Chaque contrainte `among` associe le nombre de jetons "début" dans la mesure i à la variable o_i. Ensuite, à chaque groupe de b mesures, la contrainte `alldifferent` est appliquée sur les variables O. Enfin, la contrainte `oracle` est déclarée sur la variable courante x_t à générer où p correspond aux probabilités marginales provenant du CMT au pas de temps t. Les variables x_1 à x_{t-1} sont déjà assignées aux valeurs de jeton générées.

5.6.1 Contrainte `oracle` (non pondérée)

Nous expérimentons d'abord le comportement avec la contrainte `oracle` non pondérée (poids à 1,0). Le tableau 5.9 montre que les résultats sont de moins en moins similaires au jeu de données de test plus le nombre de mesures contraintes augmente. En effet, lorsque la contrainte est appliquée sur quatre mesures, le nombre de motifs rythmiques différents pour une mesure est de 3450 avec une JSD de 0,248. Ces métriques augmentent jusqu'à 4706 et 0,367 lorsque la contrainte est appliquée sur huit mesures. Ainsi, les mélodies générées sont considérablement différentes rythmiquement du jeu de données de test. En effet, dans seulement 6% des exemplaires du jeu de données de test, la contrainte est naturellement

Tableau 5.9 Métriques rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une contrainte `oracle` non pondérée.

	Nombre mesures impliquées dans la contrainte					Test
	4	5	6	7	8	
Nb motifs rythmiques diff mesure	3450	3812	4078	4399	4706	1424
Divergence Jensen-Shannon	0,248	0,272	0,299	0,334	0,367	-

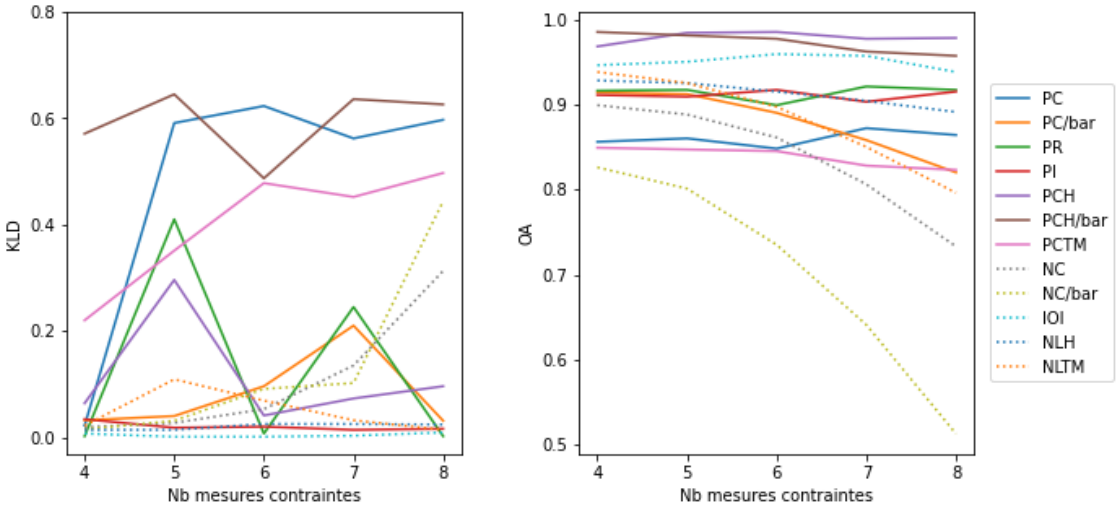

Figure 5.8 Métriques du MGEval *framework* des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une contrainte `oracle` non pondérée.

respectée lorsqu'appliquée sur quatre mesures, et moins de 0,2% sur huit mesures. La figure 5.8 montre la progression de la KLD et du OA du *framework* MGEval plus le nombre de mesures contraintes augmente. Il est possible de remarquer que les métriques NC et NC/bar ont beaucoup augmenté en KLD et diminué en OA. Ceci peut être attendu, car le modèle doit générer des mesures contenant plus de quatre notes engendrant ainsi plus de notes en général que le jeu de données de test (comme expliqué à la section 5.6). Ainsi, cela permet de confirmer que les résultats sont de moins en moins similaires au jeu de données de test plus le nombre de mesures contraintes augmente. Il est aussi possible de voir que la KLD augmente considérablement pour les métriques PC et PCTM. En effet, lorsque le rythme change, indirectement, les hauteurs de note aussi. Cependant, leur OA ne semble pas tant être affecté. Dans l'article du MGEval *framework* [33], les auteurs expliquent que la KLD et l'OA sont des métriques complémentaires. L'OA ne permet pas de bien capturer la différence entre deux PDFs lorsque leur moyenne est semblable, mais que leur forme est différente (kurtosis et asymétrie). D'une autre part, la KLD ne permet pas de bien discerner deux

PDFs lorsqu'elles ont une forme semblable, mais une moyenne différente. Ainsi, les KLDs du PC et du PCTM semblent montrer une différence dans la forme des PDFs, plus le nombre de mesures contraintes augmente.

Les *heat maps* de la figure 5.9 montrent que le *problème de procrastination* est aussi présent pour cette contrainte. Effectivement, on retrouve les zones foncées en bas à droite de chaque *heat map*. Dans ce contexte, vers la fin de la génération d'une mesure, le modèle est forcé de générer des jetons "début" afin de ne pas avoir le même nombre de notes que les mesures précédentes. Le ratio de mesures ayant un jeton "début" comme dernier pas de temps est de 4% dans le jeu de données de test, alors que dans le jeu de données généré, le ratio atteint 27% à 43% lorsque la contrainte est appliquée sur quatre à huit mesures.

Figure 5.9 *Heat maps* des tops 10 des motifs rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une contrainte `oracle` non pondérée.

5.6.2 Décroissance du poids de la contrainte `oracle` après chaque jeton

Compte tenu que la contrainte d'avoir un nombre différent de notes à chaque mesure est aussi affectée par le *problème de procrastination*, nous appliquons donc notre méthode consistant à une décroissance géométrique du poids de la contrainte `oracle`.

Tableau 5.10 Métriques rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,9943$ du poids de l'`oracle`.

	Nombre mesures impliquées dans la contrainte					Test
	4	5	6	7	8	
Nb motifs rythmiques diff mesure	5205	5983	7151	8477	9808	1424
Divergence Jensen-Shannon	0,266	0,298	0,331	0,371	0,403	-

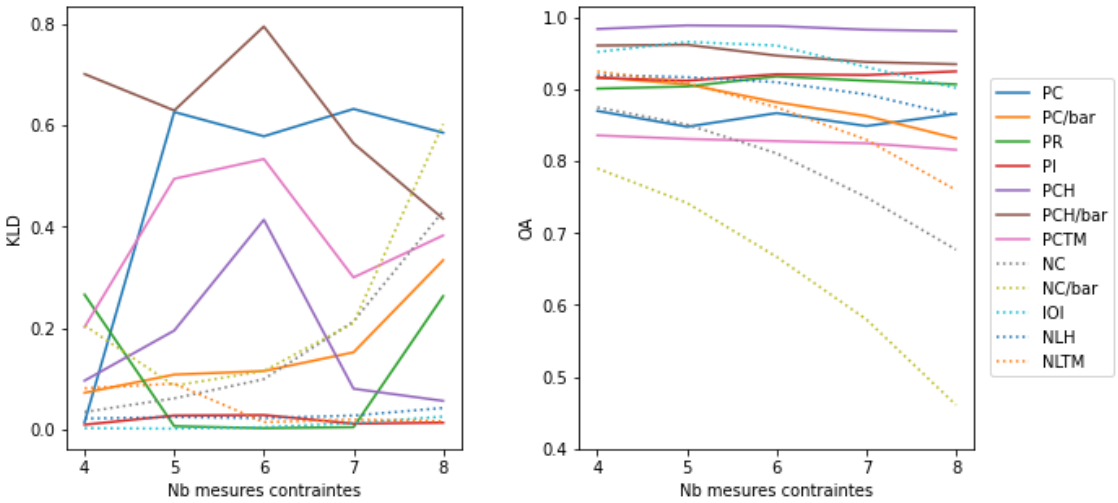

Figure 5.10 Métriques du MGEval *framework* des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,9943$ du poids de l'`oracle`.

La figure 5.11 montrent des résultats avec une raison $r = 0,9943$. Cette valeur r a été déterminée de la même manière qu'à la section 5.5.6, soit pour qu'après quatre mesures générées (64 jetons), le poids atteigne une valeur d'environ 0,7. Le *problème de procrastination* a grandement été mitigé vu l'absence des régions foncées comparativement aux expériences précédentes (figure 5.11). Le ratio de mesures ayant un jeton "début" comme dernier pas de temps atteint 23% à 32% lorsque la contrainte est appliquée sur quatre à huit mesures. En revanche, un compromis avec la similarité du jeu de données de test a dû être fait pour obtenir

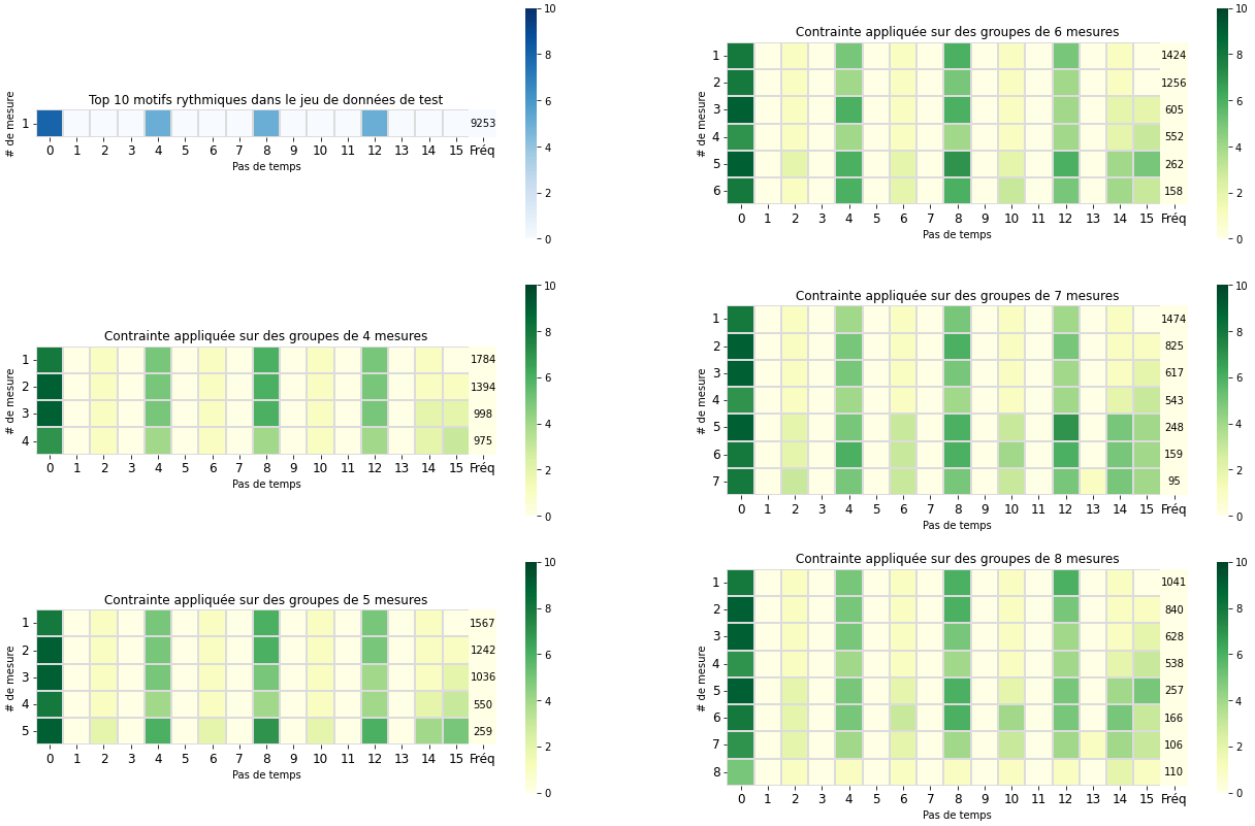

Figure 5.11 *Heat maps* des tops 10 des motifs rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,9943$ du poids de l'`oracle`.

une telle mitigation. Effectivement, le tableau 5.10 montre une grande augmentation dans le nombre de motifs rythmiques et de la JSD. À la figure 5.10, les métriques NC et NC/bar sont encore plus impactées. En effet, en ce qui concerne la KLD, elles atteignent plus de 0,4 et 0,6 respectivement comparativement à 0,3 et 0,4. Pour l'OA, celles-ci diminuent davantage rendu à huit mesures contraintes. Par conséquent, les mélodies générées sont davantage différentes du jeu de données de test. Encore une fois, cette différence peut être expliquée par le fait que le poids de l'`oracle` décroît lors de la génération. Dès lors, la génération est de moins en moins influencée par le style appris par CMT. Bref, la difficulté demeure toujours à trouver un équilibre convenable entre la mitigation du *problème de procrastination* et de la similarité avec le jeu de données de test. D'autres valeurs de raison r testées sont présentées à l'annexe A.1.1.

5.7 Occurrence de chaque note de la gamme

Pour cette expérience, nous nous intéressons maintenant à la génération des hauteurs de note. En effet, le nombre de valeurs possibles est beaucoup plus grand que celui du jeton de rythme. Nous voulons donc étudier un tel impact sur notre méthode. Comme mentionné à la section 5.1, CMT génère des mélodies en *do* majeur ou en *la* mineur (les deux gammes contiennent exactement les mêmes hauteurs de note). Par conséquent, pour cette expérience, nous avons implémenté un modèle CP avec MiniCPBP qui consiste à avoir au moins une occurrence de chaque note de la gamme (*do*, *ré*, *mi*, *fa*, *sol*, *la* et *si*). Ainsi, c'est une contrainte qui affecte seulement les hauteurs de note. Cependant, pour garantir que les mélodies générées aient au moins sept notes (une pour chaque note de la gamme), nous avons aussi ajouté une contrainte sur le rythme qui consiste à avoir au moins huit notes parmi les mesures contraintes. Cette contrainte sur le rythme ne devrait pas trop affecter la génération des mélodies. Effectivement, CMT est déjà très susceptible de générer plus de huit notes sur six mesures ou plus. Ainsi, différentes expériences ont été faites en imposant ces contraintes sur des groupes de six à huit mesures. Contrairement à la contrainte de la section 5.5 et 5.6, ces contraintes ont été appliquées sur seulement le premier groupe de mesures. Par exemple, lorsque les contraintes sont appliquées sur un groupe de six mesures, seulement les mesures 1,2,3,4,5,6 sont contraintes, et pas les mesures 7,8.

De façon formelle, nous définissons $CSP_{HdeN}(X \cup O, D, C)$ avec $X = \{x_1, \ldots, x_{128}\}$ l'ensemble de variables représentant chacun des 128 jetons de la séquence de hauteur de notes. $D(x_i) = \{0, \ldots, 49\}$ pour chaque $x_i \in X$, car il y a 50 valeurs possibles pour les jetons de hauteur de note comme mentionné à la section 4.1. Les variables $O = \{o_1, \ldots, o_{12}\}$ représentent le nombre d'occurrences de chacune des 12 classes de hauteur de note dans le groupe de b

mesures. $D(o_i) = \{1,\ldots, n\}$ si la classe de hauteur de note i fait partie de la gamme, sinon $D(o_i) = \{0,\ldots, n\}$ où n est le nombre de jetons "début" dans le groupe de b mesures.

Les contraintes C sont les suivantes :

$$\texttt{among}(\{x_1, \ldots, x_{b\cdot 16}\}, HNIdx_i, o_i) \quad \forall i \in \{1, \ldots, 12\} \tag{5.15}$$

$$\texttt{sum}(O) \leq n \tag{5.16}$$

$$\texttt{oracle}(x_t, D(x_t), p) \tag{5.17}$$

Chaque contrainte `among` associe le nombre d'occurrences de la classe de hauteur de note i dans le groupe de b mesures à la variable o_i. $HNIdx_i$ représente l'ensemble des valeurs de jeton (parmi les 50) correspondant à la classe de hauteur de note i. La contrainte 5.16 permet d'ajouter une vision globale aux contraintes `among`. Enfin, la contrainte `oracle` est déclarée sur la variable courante x_t à générer où p correspond aux probabilités marginales provenant du CMT au pas de temps t. Les variables x_1 à x_{t-1} sont déjà assignées aux valeurs de jeton générées.

Pour la contrainte sur le rythme, nous définissons $CSP_{rythme3}(X, D, C)$ avec $X = \{x_1, \ldots, x_{128}\}$ l'ensemble de variables représentant chacun des 128 jetons de la séquence de rythme. $D(x_i)$ = {0, 1, 2} pour chaque $x_i \in X$.

La contrainte C est la suivante :

$$\texttt{atleast}(\{x_1, \ldots, x_{b\cdot 16}\}, \text{“jeton\_début”}, 8) \tag{5.18}$$

$$\texttt{oracle}(x_t, D(x_t), p) \tag{5.19}$$

où b représente le nombre de mesures affectées par la contrainte. Les variables x_1 à x_{t-1} sont déjà assignées aux valeurs de jeton générées. Il est important de noter que le poids de la contrainte `oracle` pour ce modèle CP sur le rythme restera fixe à 1,0 pour toutes les expériences dans les sections suivantes.

5.7.1 Contrainte `oracle` (non pondérée)

Comme à la section 5.6.1, les probabilités marginales provenant de CMT sont données au modèle CP à travers la contrainte `oracle`, et des itérations de BP sont faites. Le poids de la contrainte `oracle` est fixé à 1,0, soit la valeur par défaut, tout au long de la génération.

Le tableau 5.11 montre l'effet d'imposer la contrainte d'occurrence de chaque note de la gamme sur le *chord tone ratio*. Les métriques sont devenues plus basses comparativement à ceux du jeu de données de test. La contrainte imposée est naturellement satisfaite entre 10% et 16% dans les exemplaires de test lorsque calculée sur six à huit mesures. De plus, les *chord tone ratios* ne semblent pas fortement être impactés par l'augmentation du nombre de mesures contraintes. En effet, les écarts entre la valeur maximale et minimale parmi les *chord tone ratios* du jeu de données généré sont seulement de 0,6% et 1,4% pour la métrique en général et sur le premier temps respectivement.

Tableau 5.11 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une contrainte `oracle` non pondérée.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,657	0,660	0,663	0,719
Chord Tone Ratio (1er temps)	0,742	0,752	0,738	0,791

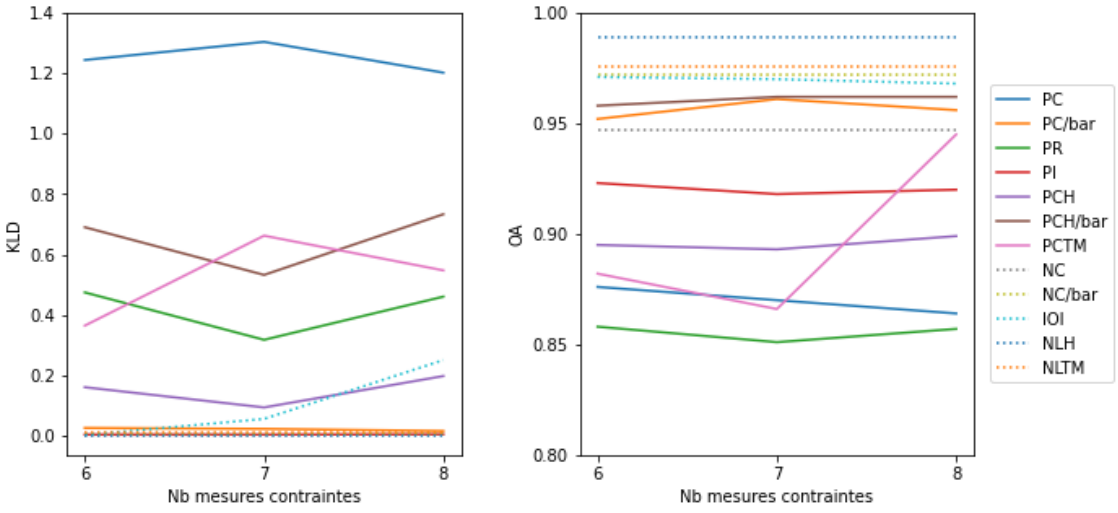

Figure 5.12 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une contrainte `oracle` non pondérée.

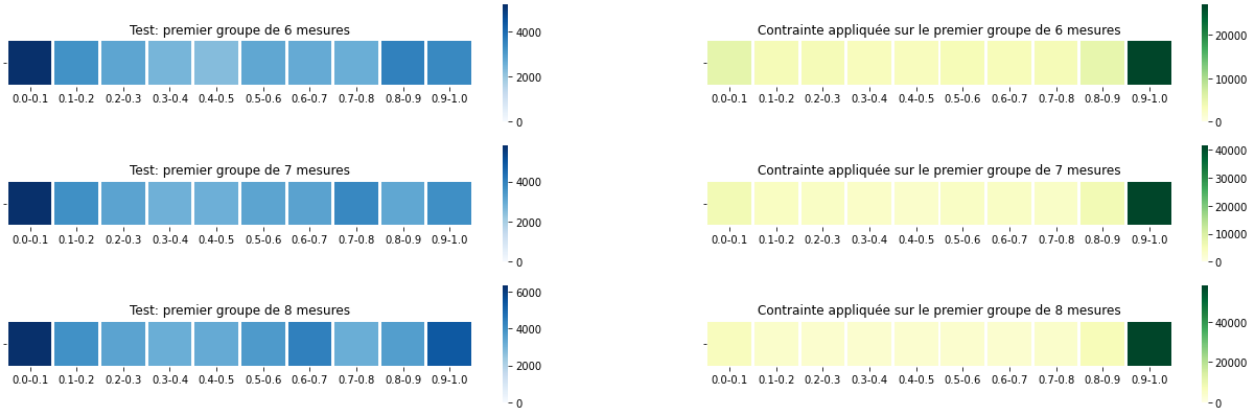

Figure 5.13 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une contrainte `oracle` non pondérée.

Les métriques du *framework* MGEval à la figure 5.12 permettent aussi de supporter cette conclusion. Effectivement, les lignes des métriques restent assez constantes, plus le nombre de mesures contraintes augmente, à l'exception du PCTM. De plus, compte tenu de la contrainte imposée, il n'est pas étonnant que pour les métriques sur les hauteurs de note, leur KLD soit parmi les plus hautes et leur OA, parmi les plus bas. En effet, il est pensable que le modèle produise des hauteurs de note qui ne générerait probablement pas sans la contrainte imposée. Ainsi, les générations des autres hauteurs de note sont aussi affectées.

Pour continuer, la figure 5.13 montre des *heat maps* représentant la fréquence d'une première occurrence d'une classe de hauteur de note de la gamme à des emplacements donnés. L'emplacement d'une note est normalisé où 0 signifie que la note est la première de la mélodie et 1, la dernière. De plus, afin de mieux visualiser le comportement des notes à la fin des mélodies, chaque compte est divisé par la probabilité d'être une première occurrence. En effet, il est beaucoup plus probable d'avoir une première occurrence de note au début de la mélodie. Plus précisément, nous faisons l'hypothèse que la probabilité de générer une des 12 classes de hauteur de note i est uniforme à chaque pas de temps, soit $\frac{1}{12}$. La probabilité d'une première occurrence de la classe de hauteur de note i au $k$ième jeton "début" est donnée par : $(1 - \frac{1}{12})^{(k-1)}(\frac{1}{12})$. En effet, pour chacun des $k - 1$ jetons "début" précédent, la classe de hauteur de note i n'a pas été générée, et on la génère au $k$ième jeton. Ainsi, pour la probabilité de n'importe quelle première occurrence des 12 classes de hauteur de note, on somme sur les 12 classes : $(1 - \frac{1}{12})^{(k-1)}$. Par conséquent, une première occurrence d'une note vers la fin d'une mélodie comptera pour plus, car il est moins probable d'en avoir une à cet emplacement comparativement à une première occurrence en début de mélodie. Sur les *heat*

maps à la figure 5.13, il est possible de remarquer les zones foncées à la dernière case à droite. Ce sont ainsi d'autres manifestations du *problème de procrastination*. Effectivement, vers la fin, le modèle n'a plus d'autre choix que de générer les notes manquantes dans la gamme pour respecter la contrainte d'en avoir au moins une occurrence. Le *chord tone ratio* sur le dernier accord est de 77% dans le jeu de données de test. Dans le jeu de données généré, il varie de 45,4% à 46,2% lorsque la contrainte est appliquée sur six à huit mesures. Encore une fois, un placement plus uniforme est souhaité.

5.7.2 Décroissance du poids de la contrainte `oracle` après chaque jeton

Pour mitiger le *problème de procrastination*, nous employons de nouveau notre méthode afin de décroître le poids de l'`oracle`. La figure 5.15 montre les *heat maps* de première occurrence en utilisant une raison $r = 0,985$ (valeur toujours basée selon le poids après quatre mesures, ici 0,4). Il est possible de remarquer qu'encore une fois, le *problème de procrastination* a notablement été mitigé. En effet, les emplacements des premières occurrences des notes de la gamme sont plus uniformes à travers la mélodie. Le *chord tone ratio* sur le dernier accord varie maintenant de 54,7% à 60,3% lorsque la contrainte est appliquée sur six à huit mesures. Cependant, cette mitigation vient aussi avec un compromis avec la similarité du jeu de données de test. En effet, le tableau 5.12 montre que les *chord tone ratios* ont davantage diminués, et s'éloignent ainsi du jeu de données de test. Par contre, à la figure 5.14, les métriques du *framework* MGEval ne semblent pas trop différer de celles avec une contrainte `oracle` non pondérée. En effet, quelques métriques sont légèrement moins bonnes (ex. PCH/bar, PCTM), alors que d'autres sont légèrement meilleures (ex. PC, PR, PI). D'autres valeurs de raison r testées sont présentées à l'annexe A.2.1.

Tableau 5.12 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique $r = 0,985$ du poids de l'`oracle`.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,630	0,621	0,611	0,719
Chord Tone Ratio (1er temps)	0,700	0,686	0,671	0,791

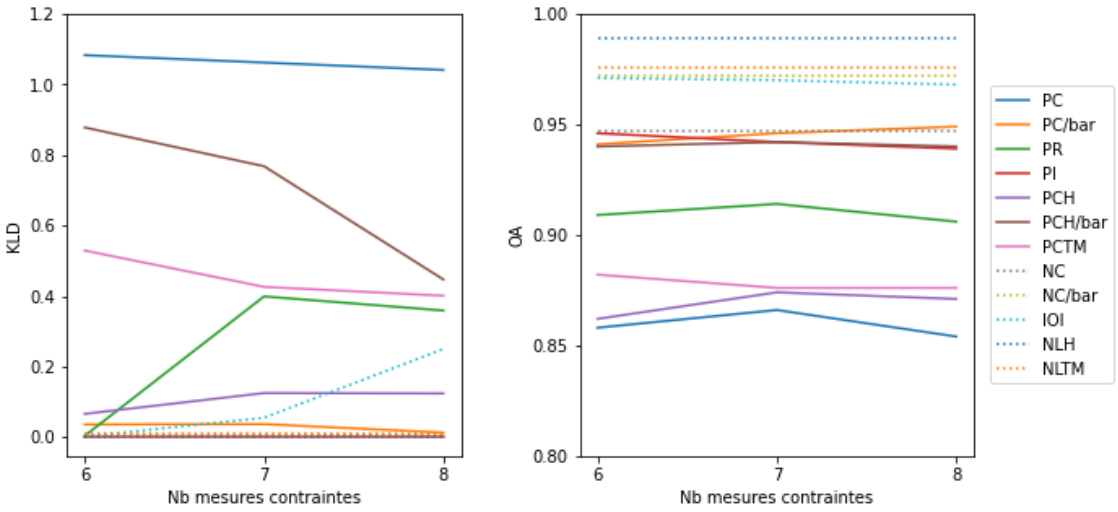

Figure 5.14 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique $r = 0,985$ du poids de l'`oracle`.

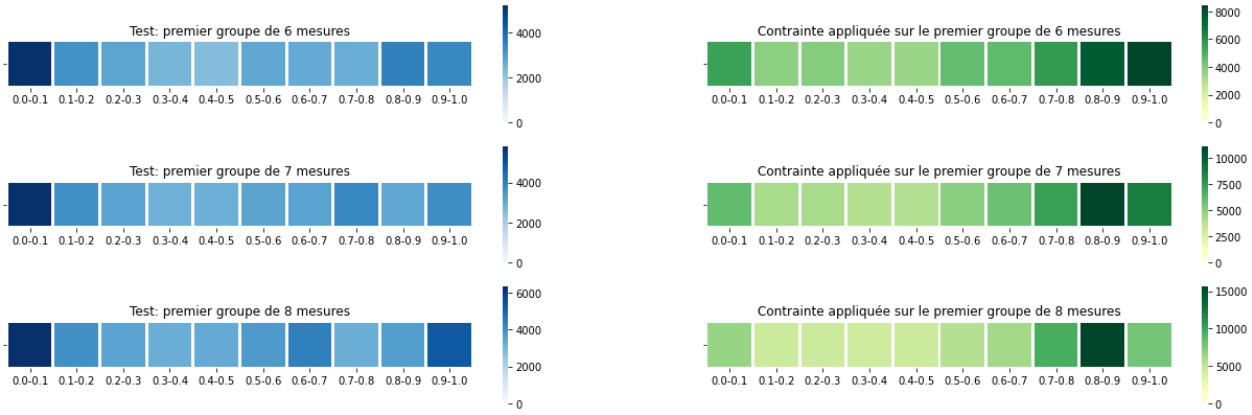

Figure 5.15 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une décroissance géométrique $r = 0,985$ du poids de l'`oracle`.

5.7.3 Décroissance du poids de la contrainte `oracle` après chaque jeton “début”

Comme mentionné à la section 4.1, la séquence de rythme est d'abord entièrement générée avant celle de hauteur de note. De ce fait, avant même de commencer à générer la séquence de hauteur de note, certains jetons sont déjà fixés au préalable. En effet, lorsqu'un jeton de rythme au temps t est “maintien” (ou “silence”), alors le jeton de hauteur de note au même temps doit aussi être “maintien” (ou “silence”). Ainsi, le nombre de jetons “début” dans la séquence de rythme correspond en fait au “vrai” nombre de jetons à générer dans la séquence de hauteur de note. Par conséquent, pour la séquence de hauteur de note, il semble plus approprié de décroître le poids après la génération d'un jeton qui n'est pas “maintien” ou “silence”. En continuant vers cette idée, comme le nombre de notes générées pour chaque mélodie est différent, il paraît moins approprié d'utiliser une même raison r fixe pour toutes les mélodies. Effectivement, le poids de la contrainte `oracle` risque de devenir très petit pour les mélodies contenant plus de notes. À la place, la valeur du poids de l'`oracle` minimale désirée est donnée. Par la suite, comme le nombre de notes total pour chaque mélodie est différent, la raison r est calculée individuellement pour chaque mélodie selon le nombre de notes afin que le poids de la contrainte `oracle` arrive à la valeur minimale donnée une fois rendu exactement au dernier “vrai” jeton à générer (qui n'est pas “maintien” ou “silence”).

Pour une valeur minimale donnée de 0,2, les résultats sont donnés au tableau 5.13, à la figure 5.16 et 5.17. Il est possible de remarquer que les résultats sont très similaires à ceux générés avec une décroissance après chaque jeton. Ainsi, il ne semble pas avoir de grande différence entre les deux méthodes lorsqu'une raison r appropriée selon chaque méthode est utilisée. Par contre, la méthode présentée dans cette section est spécifique au modèle CMT, alors que la méthode diminuant le poids après chaque jeton est plus générale. Par conséquent, une décroissance après chaque jeton reste une meilleure méthode. D'autres valeurs minimales testées sont présentées à l'annexe A.2.2.

Tableau 5.13 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. La valeur minimale donnée est 0,2.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,623	0,622	0,619	0,719
Chord Tone Ratio (1er temps)	0,692	0,688	0,679	0,791

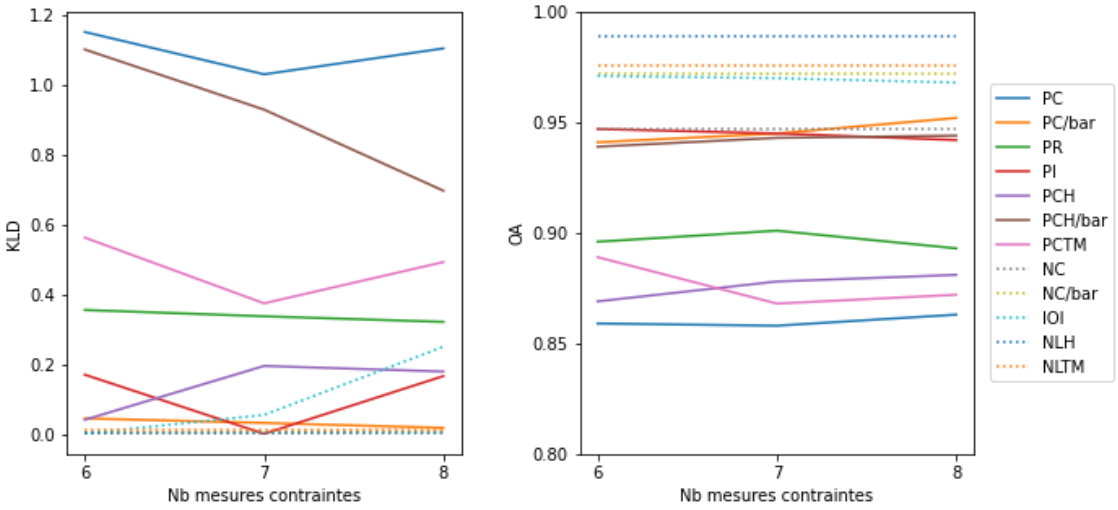

Figure 5.16 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. La valeur minimale donnée est 0,2.

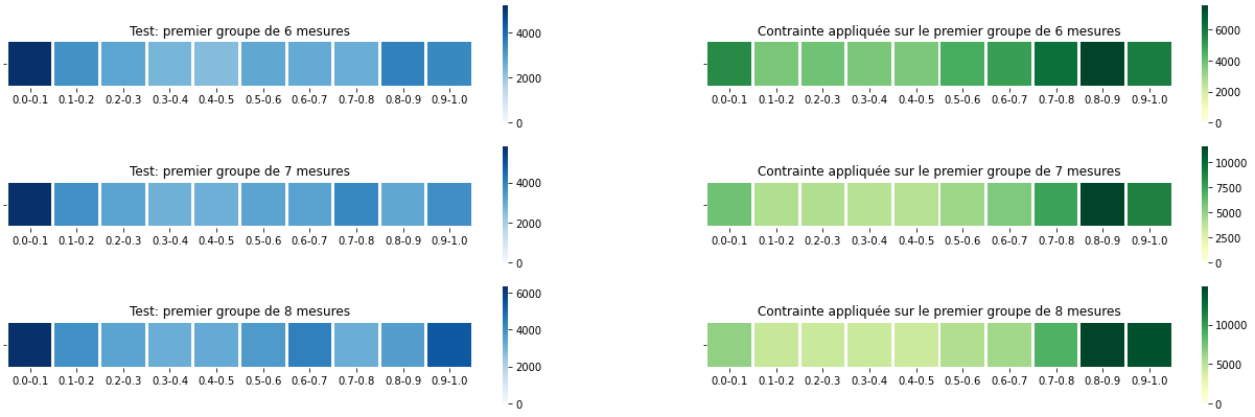

Figure 5.17 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. Valeur minimale donnée est 0,2.

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Les séquences générées par des modèles génératifs ont souvent de la difficulté à respecter une structure. En effet, la plupart des modèles de génération de séquences ne sont pas faits pour directement contrôler la séquence de sortie. Même lorsque la structure se retrouve dans le jeu de données d'entraînement, en général, il n'y a aucune garantie que les modèles de réseaux de neurones puissent effectivement l'apprendre. Ainsi, la structure doit parfois être explicitement exposée au modèle. D'une autre part, le contrôle des séquences peut aussi être utile afin d'imposer des contraintes qui ne sont pas nécessairement présentes dans le jeu de données.

Dans ce mémoire, nous avons proposé une méthode permettant d'ajouter des contraintes aux séquences de modèle d'apprentissage automatique lors de la phase d'inférence/génération. Nous nous sommes concentré sur le domaine de la génération de musique. Effectivement, il est possible de représenter la musique sous forme de séquence. De plus, pouvoir contrôler la musique générée est un problème courant dans ce domaine. Nous sommes partis du CMT, soit un modèle de l'état de l'art permettant de générer une mélodie selon une suite d'accords donnée. Notre méthode consistait à exprimer les contraintes sous la forme d'un modèle de CP. À chaque pas de temps, les probabilités marginales provenant du CMT étaient données au solveur MiniCPBP par l'entremise de la contrainte `oracle`. Puis, le solveur effectuait des itérations de BP. Grâce à la contrainte `oracle`, les probabilités marginales résultantes du processus de BP tenaient compte de respect des contraintes modélisées et aussi du style de musique appris par CMT. Afin de mitiger le *problème de procrastination* pour des contraintes à long terme, le poids de la contrainte `oracle` a été diminué de façon géométrique après la génération de chaque jeton.

Des expériences ont été faites en contraignant le rythme et les hauteurs de note des mélodies générées. Notre méthode a permis d'obtenir un équilibre raisonnable entre la mitigation du *problème de procrastination* et la similarité avec le jeu de données de test.

6.2 Limitations

La méthode proposée dans ce mémoire a certaines limitations. Tout d'abord, pour des contraintes à long terme, si le nombre de mesures contraintes augmente considérablement, dépendamment de la valeur de la raison r, le poids de l'oracle peut devenir très petit ($< 0,01$).

Rendu à ce point, les probabilités marginales provenant du CMT n'ont presque plus d'impact sur la génération. Ainsi, un certain seuil pourrait être mis en place afin de s'assurer que CMT ait toujours un minimum d'influence durant la génération. Une autre technique serait de donner une raison r afin d'atteindre une valeur minimale donnée comme à la section 5.7.3. D'un autre côté, dans le domaine de la génération de musique, il faudrait juger à quel point il est réaliste que des contraintes à long terme, telles que celles de nos expériences, soient imposées sur beaucoup de mesures (ex., 16).

Pour continuer, le travail de ce mémoire aurait grandement bénéficié de plus d'expertise en théorie de la musique. Bien que nous avons les connaissances de base, un expert aurait possiblement pu aider à trouver des contraintes plus réalistes pour les expériences. De plus, dans nos évaluations, aucun test d'écoute n'a été fait. De ce fait, une expertise en théorie de la musique aurait pu aider à qualitativement évaluer la qualité des mélodies générées. En effet, il est pertinent de pouvoir contrôler les mélodies. Toutefois, il est important de pouvoir le faire tout en gardant une qualité satisfaisante [19]. Par conséquent, il est difficile d'évaluer la qualité des mélodies seulement avec des métriques quantitatives.

Pour poursuivre, dans ce mémoire, nous nous sommes concentrés sur les contraintes à long terme, et comment mitiger le *problème de procrastination*. Cependant, avec notre méthode, il est aussi possible d'imposer d'autres types de contrainte qui ne sont pas nécessairement à long terme. Un exemple serait d'avoir une certaine hauteur de note à un temps spécifique. Dans ce cas-ci, le modèle CP forcerait simplement cette hauteur de note (en mettant les probabilités marginales des autres valeurs à zéro) une fois rendu au pas de temps concerné. Cependant, cette façon correspond exactement à la méthode naïve motivant la création du modèle Anticipation-RNN présenté à la section 3. En effet, l'apparition soudaine de la note imposée viendrait perturber la distribution des notes générées précédemment. De ce fait, notre méthode ne générera pas les notes précédentes en tenant compte de la note imposée qui arrive dans le cas où de simples contraintes unaires sont imposées.

6.3 Améliorations et travaux futurs

Le jeu de données utilisé pour les expériences est le EWLD contenant plusieurs styles de musique. Ainsi, une amélioration serait de recourir à un jeu de donnée avec un seul style de musique. De ce fait, les résultats seraient plus faciles à juger qualitativement afin de voir si le style est toujours respecté. Cependant, il est difficile de trouver un tel jeu de données où chaque exemplaire contient explicitement une mélodie et une suite d'accords pouvant être séparé. En effet, c'est ce que CMT requiert pour être entraîné.

Comme mentionné à la section 5.2, lorsqu'un modèle CP est trop complexe, il est possible qu'une séquence partiellement générée ne puisse pas aboutir à une séquence respectant les contraintes. Un exemple de modèle CP serait que chaque mesure doit contenir un nombre différent de notes et que la dernière mesure contienne deux fois plus de notes que l'avant-dernière mesure. Dans cet exemple, une partie des mélodies ne pouvaient pas arriver à une solution réalisable. Dans ce cas, une simple solution serait de réinitialiser les séquences problématiques et recommencer la génération. Une autre solution serait d'utiliser le mécanisme de marche arrière (*backtracking*) de la CP. Pour ce faire, la génération de la séquence se ferait maintenant à travers la recherche de solutions de la CP. Plus précisément, le branchement de variable serait exécuté en ordre lexicographique sur les variables X représentant les jetons. Ensuite, la valeur choisie serait échantillonnée selon les probabilités marginales résultantes de la BP. Ainsi, à chaque branchement, les marginales du CMT devront être obtenues pour dynamiquement poser la contrainte `oracle` dans le modèle CP. De cette façon, lors d'une marche arrière, la contrainte `oracle` serait aussi mise en marche arrière. Cette méthode permettrait de gérer les séquences partielles non prometteuses lorsque le modèle CP est complexe. Cependant, le temps d'exécution risque d'être considérablement plus long.

Pour continuer, les expériences se sont grandement concentrées sur des contraintes globales à long terme. Cependant, dans le domaine de la musique, lorsqu'on parle de structure, il est souvent question de répétition et de variation des différentes sections de la musique générée. Ainsi, il aurait été intéressant de tester notre méthode avec des contraintes impliquant ces deux aspects.

Dernièrement, comme travaux futurs, il s'agirait de montrer davantage la généralité de notre méthode. Par exemple, en expérimentant avec d'autres modèles de base que le CMT. Il serait aussi intéressant d'explorer notre méthode dans d'autres domaines d'application. Un exemple de domaine serait évidemment le NLP où les séquences générées doivent respecter certaines contraintes sémantiques ou syntaxiques. Dans le domaine musical, les contraintes imposées servaient surtout à guider la créativité du modèle d'apprentissage automatique, alors que dans le domaine du NLP, les contraintes représentent davantage des règles ou des connaissances préalables que la séquence doit respecter pour être valide.

RÉFÉRENCES

[1] K. Choi *et al.*, "Chord conditioned melody generation with transformer based decoders," *IEEE Access*, vol. 9, p. 42 071–42 080, 2021.

[2] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, p. 1877–1901, 2020.

[3] T. Karras *et al.*, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv :1710.10196*, 2017.

[4] J.-P. Briot, G. Hadjeres et F.-D. Pachet, "Deep learning techniques for music generation–a survey," *arXiv preprint arXiv :1709.01620*, 2017.

[5] C.-Z. A. Huang *et al.*, "Music transformer," *arXiv preprint arXiv :1809.04281*, 2018.

[6] D. Deutsch, S. Upadhyay et D. Roth, "A general-purpose algorithm for constrained sequential inference," dans *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, 2019, p. 482–492.

[7] S. Lattner, M. Grachten et G. Widmer, "Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints," *Journal of Creative Music Systems*, vol. 2, p. 1–31, 2018.

[8] J. Y. Lee *et al.*, "Gradient-based inference for networks with output constraints," dans *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, n°. 01, 2019, p. 4147–4154.

[9] Y. Nandwani, A. Pathak et P. Singla, "A primal dual formulation for deep learning with constraints," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] F. Pachet, P. Roy et G. Barbieri, "Finite-length markov processes with constraints," dans *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[11] I. Goodfellow, Y. Bengio et A. Courville, *Deep learning*. MIT press, 2016.

[12] S. Hochreiter et J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, n°. 8, p. 1735–1780, 1997.

[13] D. Bahdanau, K. Cho et Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv :1409.0473*, 2014.

[14] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] A. P. Parikh *et al.*, "A decomposable attention model for natural language inference," *arXiv preprint arXiv :1606.01933*, 2016.

[16] F. Rossi, P. Van Beek et T. Walsh, *Handbook of constraint programming.* Elsevier, 2006.

[17] E. Waite *et al.*, "Generating long-term structure in songs and stories," *Web blog post. Magenta*, vol. 15, n°. 4, 2016.

[18] P. Shaw, J. Uszkoreit et A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv :1803.02155*, 2018.

[19] H. Young *et al.*, "Compositional steering of music transformers," *Intelligent User Interfaces*, 2022.

[20] E. B. Zaken, S. Ravfogel et Y. Goldberg, "Bitfit : Simple parameter-efficient fine-tuning for transformer-based masked language-models," *arXiv preprint arXiv :2106.10199*, 2021.

[21] G. Hadjeres et F. Nielsen, "Anticipation-rnn : Enforcing unary constraints in sequence generation, with application to interactive music generation," *Neural Computing and Applications*, vol. 32, n°. 4, p. 995–1005, 2020.

[22] A. Roberts *et al.*, "A hierarchical latent vector model for learning long-term structure in music," dans *International conference on machine learning.* PMLR, 2018, p. 4364–4373.

[23] D. P. Kingma et M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv :1312.6114*, 2013.

[24] G. E. Hinton et R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, n°. 5786, p. 504–507, 2006.

[25] V. Punyakanok, D. Roth et W.-t. Yih, "The importance of syntactic parsing and inference in semantic role labeling," *Computational Linguistics*, vol. 34, n°. 2, p. 257–287, 2008.

[26] O. Täckström, K. Ganchev et D. Das, "Efficient inference and structured learning for semantic role labeling," *Transactions of the Association for Computational Linguistics*, vol. 3, p. 29–41, 2015.

[27] L. Michel, P. Schaus et P. Van Hentenryck, "Minicp : a lightweight solver for constraint programming," *Mathematical Programming Computation*, vol. 13, n°. 1, p. 133–184, 2021. [En ligne]. Disponible : https://doi.org/10.1007/s12532-020-00190-7

[28] G. Pesant, "From support propagation to belief propagation in constraint programming," *Journal of Artificial Intelligence Research*, vol. 66, p. 123–150, 2019.

[29] F. Simonetta *et al.*, "Symbolic music similarity through a graph-based representation," dans *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion, Wrexham, United Kingdom, September 12-14, 2018*, S. Cunningham

et R. Picking, édit. ACM, 2018, p. 26 :1–26 :7. [En ligne]. Disponible : https://doi.org/10.1145/3243274.3243301

[30] M. S. Cuthbert et C. Ariza, "Music21 : A toolkit for computer-aided musicology and symbolic music data," dans *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*, J. S. Downie et R. C. Veltkamp, édit. International Society for Music Information Retrieval, 2010, p. 637–642. [En ligne]. Disponible : http://ismir2010.ismir.net/proceedings/ismir2010-108.pdf

[31] T. Lin *et al.*, "Focal loss for dense object detection," dans *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, p. 2999–3007. [En ligne]. Disponible : https://doi.org/10.1109/ICCV.2017.324

[32] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Trans. Inf. Theory*, vol. 37, n°. 1, p. 145–151, 1991. [En ligne]. Disponible : https://doi.org/10.1109/18.61115

[33] L.-C. Yang et A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, vol. 32, n°. 9, p. 4773–4784, 2020.

[34] C. Guo *et al.*, "On calibration of modern neural networks," dans *International conference on machine learning*. PMLR, 2017, p. 1321–1330.

[35] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, n°. 3, p. 61–74, 1999.

[36] M. Mulamba *et al.*, "Hybrid classification and reasoning for image-based constraint solving," dans *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, 2020, p. 364–380.

[37] M. P. Naeini, G. Cooper et M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," dans *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

ANNEXE A RÉSULTATS D'AUTRES VALEURS DE DÉCROISSANCE TESTÉES

A.1 Nombre de notes différent par mesure

A.1.1 Décroissance après chaque jeton (section 5.6.2)

Raison $r = 0,991$

Tableau A.1 Métriques rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,991$ du poids de l'`oracle`.

	Nombre mesures impliquées dans la contrainte					Test
	4	5	6	7	8	
Nb motifs rythmiques diff mesure	6971	8196	9879	11571	13003	1424
Divergence Jensen-Shannon	0,300	0,339	0,379	0,418	0,453	-

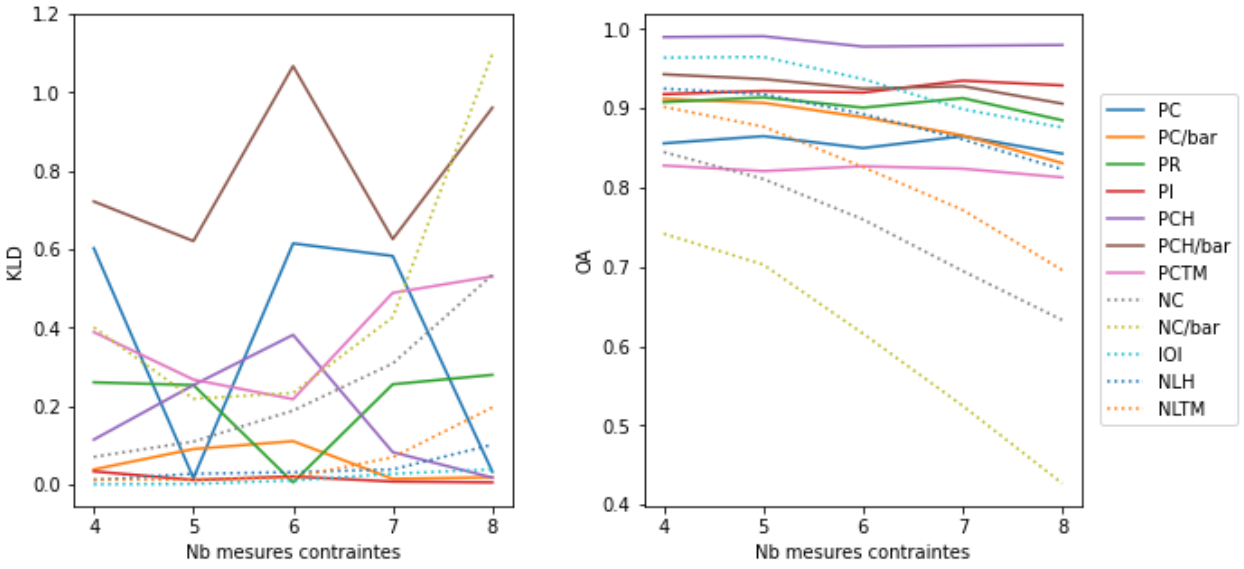

Figure A.1 Métriques du MGEval *framework* des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,991$ du poids de l'`oracle`.

Figure A.2 *Heat maps* des tops 10 des motifs rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,991$ du poids de l'`oracle`.

Raison $r = 0,9964$

Tableau A.2 Métriques rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,9964$ du poids de l'`oracle`.

	Nombre mesures impliquées dans la contrainte					Test
	4	5	6	7	8	
Nb motifs rythmiques diff mesure	4397	4969	5725	6620	7604	1424
Divergence Jensen-Shannon	0,254	0,283	0,314	0,348	0,379	-

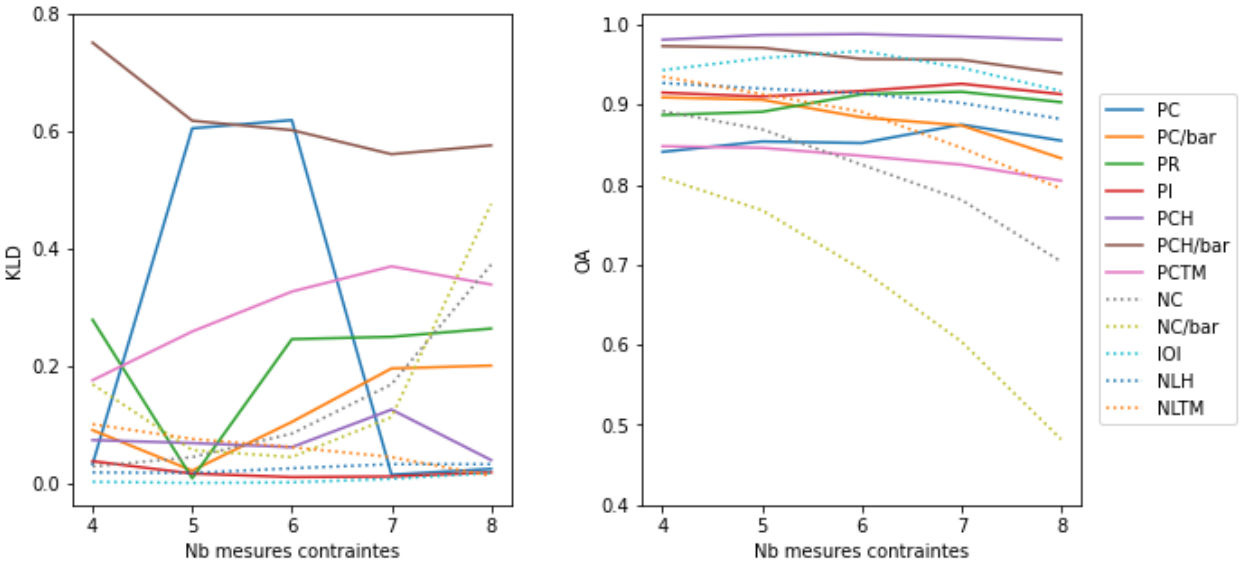

Figure A.3 Métriques du MGEval *framework* des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,9964$ du poids de l'`oracle`.

Figure A.4 *Heat maps* des tops 10 des motifs rythmiques des mélodies générées avec un nombre de notes différent par mesure en utilisant le modèle avec une décroissance géométrique $r = 0,9964$ du poids de l'`oracle`.

A.2 Occurrence de chaque note de la gamme

A.2.1 Décroissance après chaque jeton (section 5.7.2)

Raison $r = 0,981$

Tableau A.3 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique $r = 0,981$ du poids de l'`oracle`.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,622	0,612	0,602	0,719
Chord Tone Ratio (1er temps)	0,688	0,673	0,661	0,791

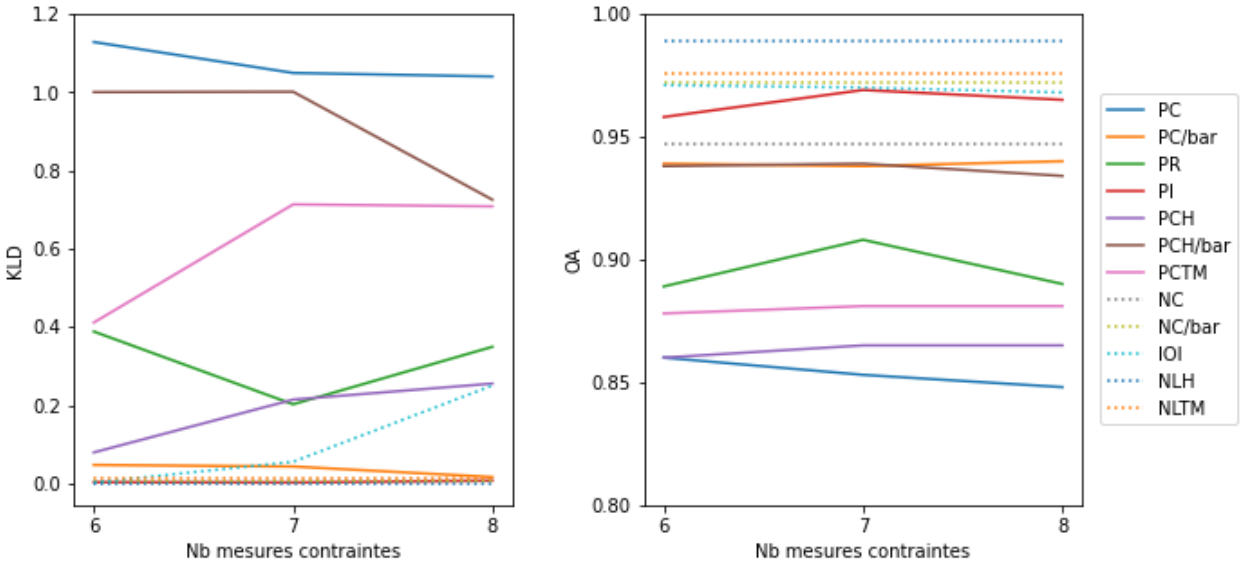

Figure A.5 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique $r = 0,981$ du poids de l'`oracle`.

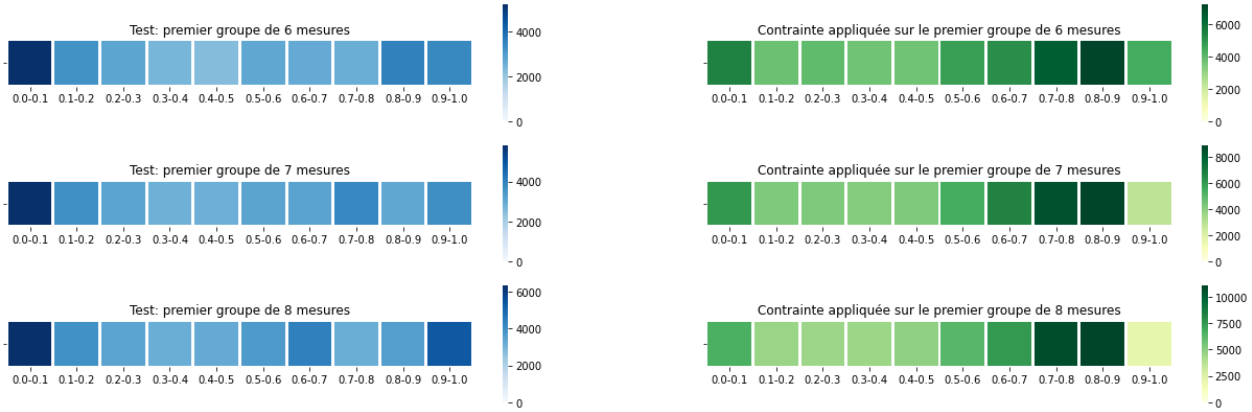

Figure A.6 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une décroissance géométrique $r = 0,981$ du poids de l'`oracle`.

Raison $r = 0,989$

Tableau A.4 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique $r = 0,989$ du poids de l'`oracle`.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,633	0,630	0,622	0,719
Chord Tone Ratio (1er temps)	0,708	0,696	0,686	0,791

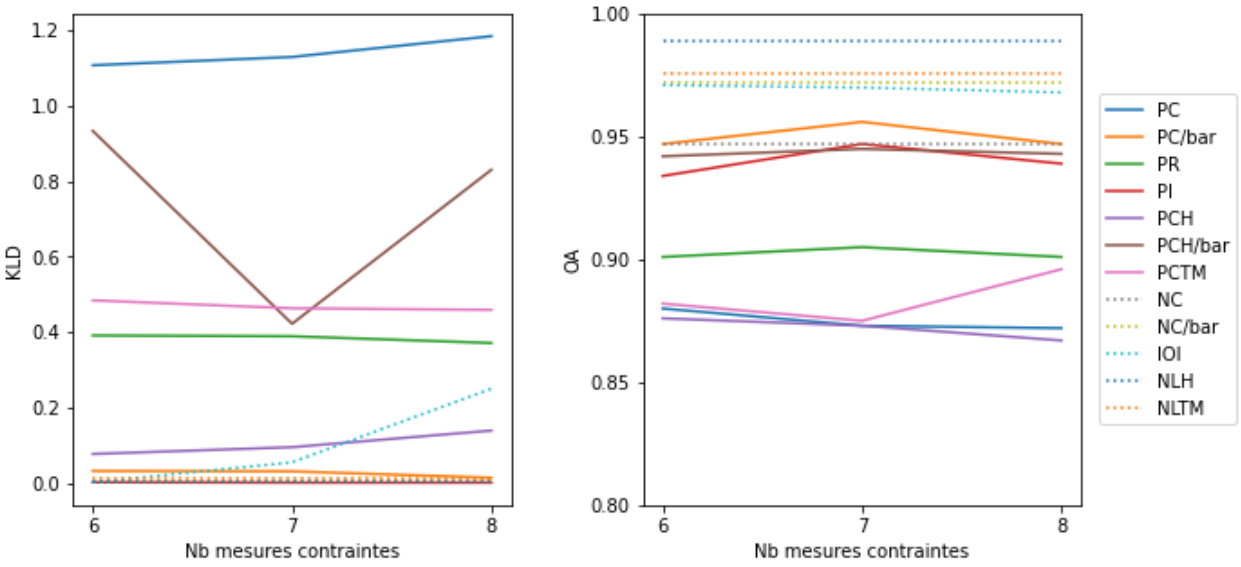

Figure A.7 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique $r = 0,989$ du poids de l'`oracle`.

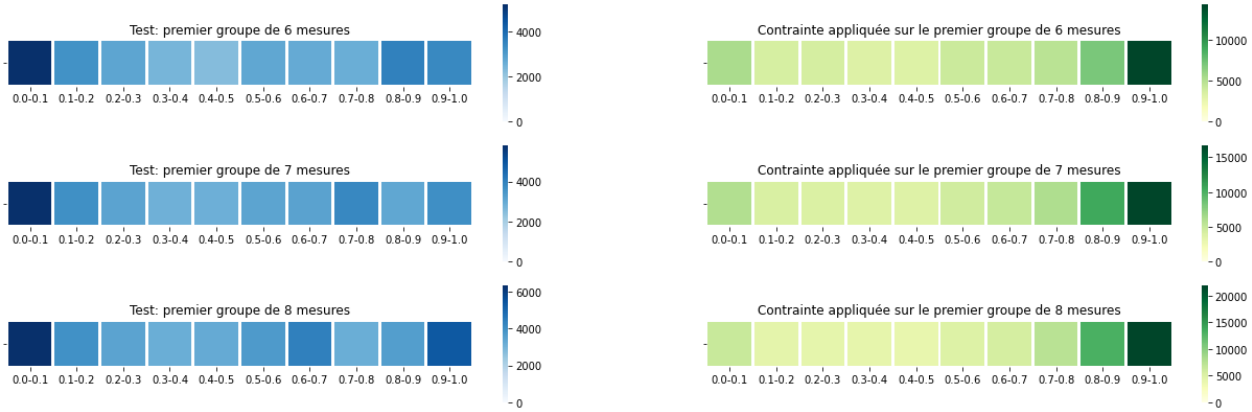

Figure A.8 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une décroissance géométrique $r = 0,989$ du poids de l'`oracle`.

A.2.2 Décroissance après chaque jeton “début” (section 5.7.3)

Valeur minimale donnée : $0,1$

Tableau A.5 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. La valeur minimale donnée est 0,1.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,616	0,610	0,606	0,719
Chord Tone Ratio (1er temps)	0,676	0,670	0,661	0,791

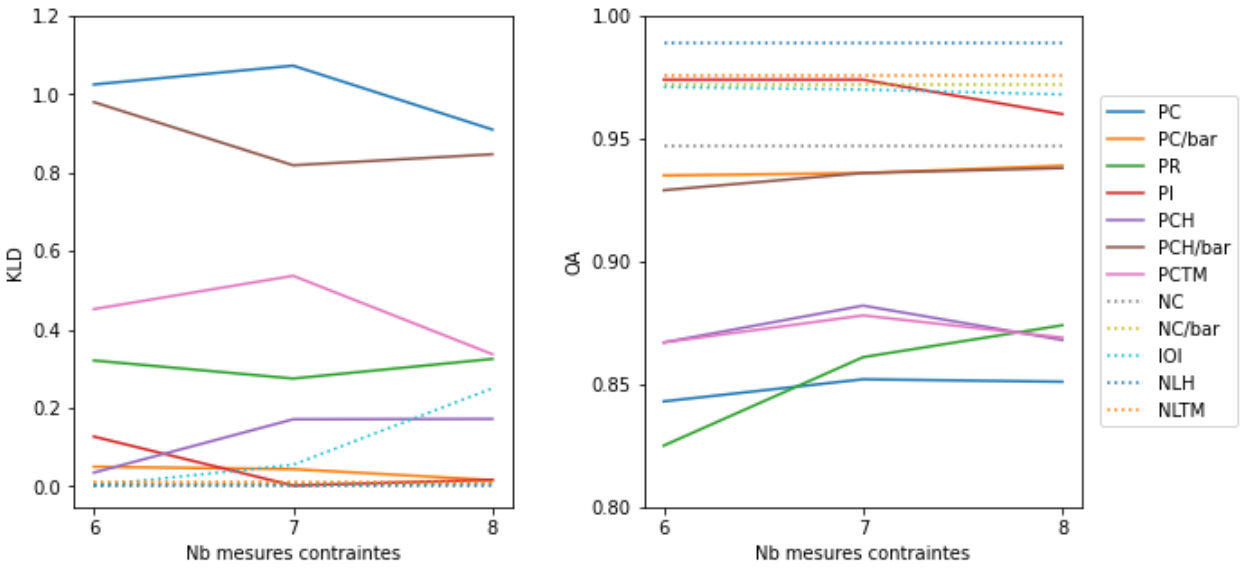

Figure A.9 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. La valeur minimale donnée est 0,1.

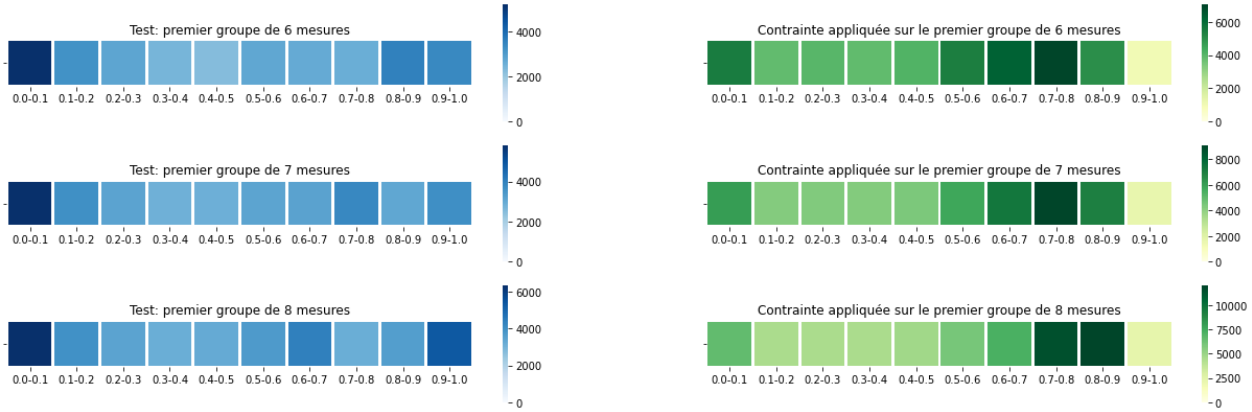

Figure A.10 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. Valeur minimale donnée est 0,1.

Valeur minimale donnée : $0,3$

Tableau A.6 Métriques de hauteur de note des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. La valeur minimale donnée est 0,3.

	Nombre mesures impliquées dans la contrainte			Test
	6	7	8	-
Chord Tone Ratio (général)	0,630	0,632	0,627	0,719
Chord Tone Ratio (1er temps)	0,700	0,702	0,693	0,791

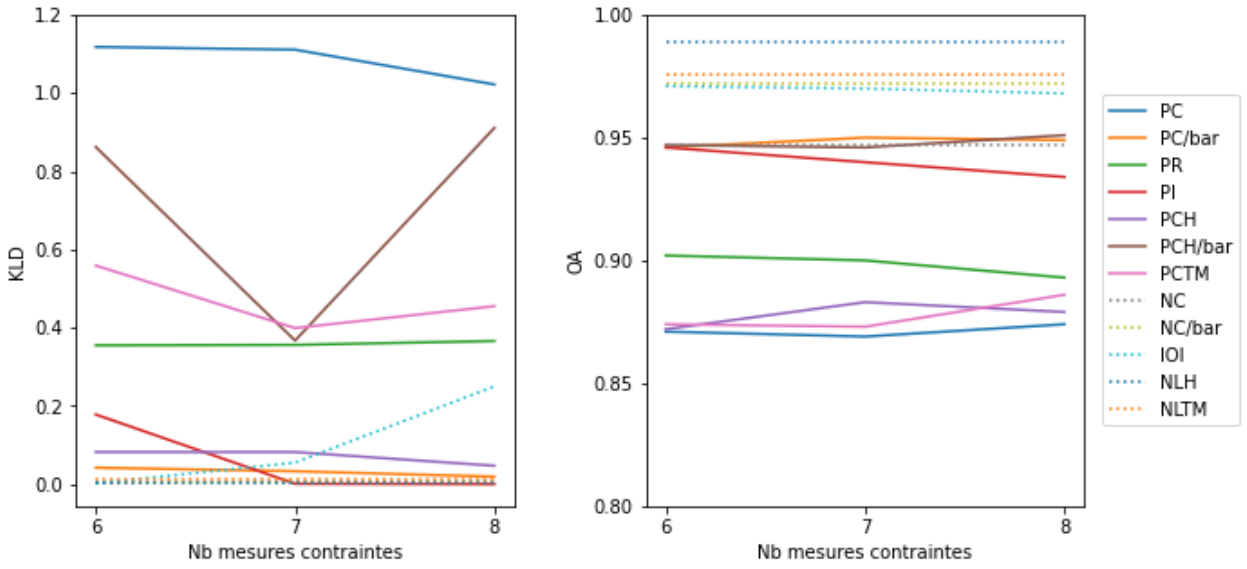

Figure A.11 Métriques du MGEval *framework* des mélodies générées avec au moins une occurrence des notes de la gamme en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. La valeur minimale donnée est 0,3.

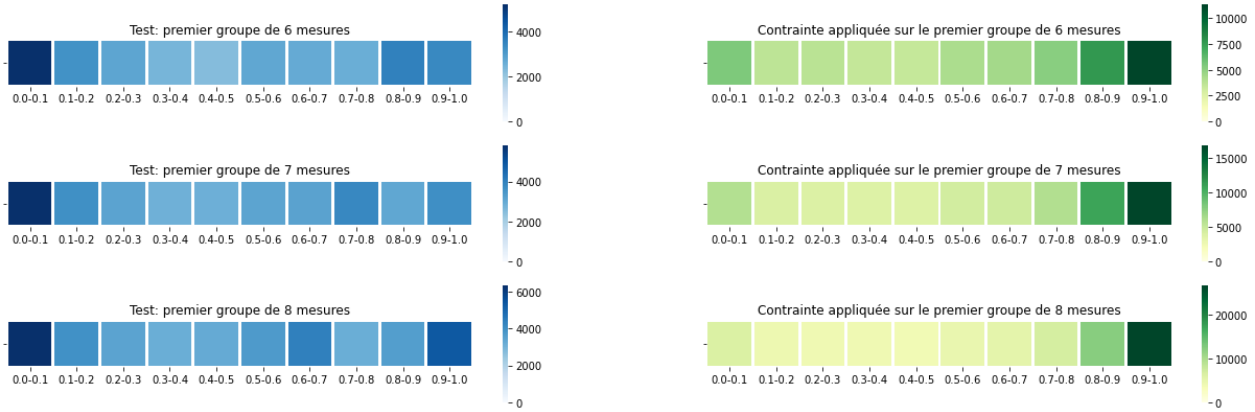

Figure A.12 *Heat maps* des placements de premières occurrences des notes de la gamme des mélodies générées en utilisant le modèle avec une décroissance géométrique du poids de l'`oracle` après chaque jeton correspondant à un début de note. Valeur minimale donnée est 0,3.