View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by PolyPublie

UNIVERSITÉ DE MONTRÉAL

GÉNÉRATION AUTOMATIQUE DE MÉLODIE PAR LA PROGRAMMATION PAR CONTRAINTES

ALEXANDRE BRIAND
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AOÛT 2018

© Alexandre Briand, 2018.

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

GÉNÉRATION AUTOMATIQUE DE MÉLODIE PAR LA PROGRAMMATION PAR CONTRAINTES

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

M. GAGNON Michel, Ph. D., président
M. PESANT Gilles, Ph. D., membre et directeur de recherche
M. BOYER François-Raymond, Ph. D., membre

REMERCIEMENTS

J'aimerais commencer par remercier mon directeur de recherche, Gilles, pour sa disponibilité, ses idées, ses conseils et son aide financière par le biais de son groupe de recherche. Avec un équilibre parfait entre direction et liberté, j'ai pris beaucoup de plaisir à mener à bien ce projet.

Bien que je n'aie pas été très présent sur place, je tiens tout de même à remercier les membres du laboratoire QUOSSEÇA pour leur agréable compagnie. Merci donc à Jinling, Mahshid, Philippe, Rachid et Samuel.

Un gros merci à mes parents et à ma copine Camille qui me soutiennent continuellement et de multiples façons dans tout ce que j'entreprends. Ce projet ne fait pas exception.

Finalement, j'aimerais adresser mes remerciements au Service des Bourses de Saint-Pierre et Miquelon ainsi qu'à l'organisme Scholarship America pour leur soutien financier.

RÉSUMÉ

La programmation par contraintes est un type de programmation déclarative, un paradigme naturellement adapté au traitement de problèmes musicaux. En effet, la composition musicale s'apparente à un processus déclaratif pendant lequel le compositeur travaille pour créer de la musique qui respecte les règles générales de l'art et les critères plus spécifiques du style adopté tout en y incorporant ses propres contraintes. Le parallèle entre cet exercice et la résolution d'un problème de satisfaction de contraintes se fait donc instinctivement. La principale difficulté se trouve au niveau de la modélisation du problème. Une pièce musicale est composée de plusieurs dimensions entre lesquelles existent beaucoup d'interactions. Il est pratiquement impossible pour un système informatique de représenter précisément toutes ces dépendances. Les systèmes de contraintes conçus pour traiter de problèmes musicaux se concentrent alors sur des dimensions en particulier.
Parmi ces problèmes, on retrouve la génération de mélodie qui concerne donc les hauteurs et les durées des notes d'une ligne mélodique accompagnée par une suite d'accords. La modélisation d'un tel problème se concentre sur une séquence de notes et ne présente donc aucun élément de polyphonie ou d'instrumentation par exemple, ce qui simplifie la situation. L'objectif de ce projet est de concevoir un système de génération automatique de mélodie selon une suite d'accords donnée qui utilise les informations d'un corpus pour guider la composition. Deux des principaux défis de ce type de problème sont l'organisation des variables et le contrôle de la structure globale de la mélodie générée. Pour relever le premier, nous avons émis l'hypothèse qu'un système structuré hiérarchiquement offrait le plus de flexibilité et permettrait donc d'exprimer les contraintes plus facilement. En ce qui concerne la structure du résultat, nous avons mis au point un algorithme de détection de patrons répétitifs basé sur des arbres des suffixes qui permet au système de répliquer les éléments de la structure d'une mélodie existante.

Notre système est donc divisé en différents types de blocs, organisés hiérarchiquement. La mélodie est composée de mesures qui contiennent des accords sous lesquels se placent les notes. Chaque bloc possède un nombre variable de notes qui doit être déterminé en premier pour pouvoir instancier les variables correspondantes. Le système fonctionne donc en deux phases de résolutions. La première assigne à chaque mesure un patron rythmique tiré du corpus, ce qui détermine à la fois le nombre de notes et leurs durées. La seconde phase fixe les hauteurs des notes de la mélodie.

Pour générer une mélodie, l'utilisateur doit choisir une pièce du corpus et fournir au sys-

tème sa suite d'accords. Pour la première phase, il doit aussi lui fournir la liste des patrons rythmiques du morceau choisi ainsi que leurs fréquences d'apparition. Pour la résolution, les variables sont traitées en ordre croissant de tailles de domaine et les valeurs sont assignées de façon aléatoire en suivant la distribution décrite par les fréquences des patrons rythmiques. L'utilisateur peut spécifier s'il veut que ces fréquences soient celles de la pièce d'origine ou celles du corpus au complet. Il peut en plus ajouter des contraintes d'égalité ou de cardinalité sur les patrons rythmiques ou le nombre de notes des mesures, ainsi que sur les durées.

La seconde phase assigne une hauteur à chaque note de la mélodie, de la première à la dernière. La résolution est guidée par une contrainte COST-REGULAR dont l'automate est construit en prenant en comptes les statistiques récoltées dans le corpus de mélodies. Pour chaque note, sa position, l'accord joué au même moment ainsi que la note précédente sont pris en compte et les statistiques fournissent une distribution pour les hauteurs possibles en fonction de ce contexte. Comme pour la première phase, l'heuristique de choix de valeurs suit cette distribution. L'utilisateur peut une nouvelle fois ajouter des contraintes d'égalité ou de cardinalité sur les hauteurs des notes.

Pendant les deux phases de résolution, le système prend en compte les contraintes issues des patrons identifiés par l'algorithme de détection et configurés par l'utilisateur.

Un sondage en ligne a permis d'évaluer les résultats du système. Dix-neuf personnes ont répondu à nos questions et cinq d'entre eux ont indiqué qu'ils étaient musiciens. Les mélodies générées ont dans l'ensemble été appréciées par les personnes interrogées. Les sections du sondage sur la qualité et la diversité des mélodies composées par le système ont données de très bons résultats. De plus, dans l'autre partie du sondage qui portait sur la capacité d'imitation du système et qui demandait aux sujets d'identifier la mélodie composée par un être humain parmi trois mélodies, plus de quarante pour cent des réponses récoltées identifient à tort un résultat du système. Ces données permettent de démontrer qu'il n'est pas nécessaire de mettre au point des nouvelles contraintes dédiées ni de spécifier explicitement des règles du domaine musical pour composer des mélodies crédibles et créatives avec la programmation par contraintes.

ABSTRACT

Constraint programming belongs to the declarative programming paradigm which is naturally suited to tackle musical problems. Musical composition can be seen as a declarative process during which the composer works to create music respecting the general and specific rules of the chosen style and also adds his own touch. The connection between this process and resolving a constraint satisfaction problem is made instinctively. The main challenge of this field is modeling the problem because of all the different dimensions which interact together in a music piece. It is virtually impossible for a computer-based system to provide a view of the same quality a human composer would have. Thus, constraint systems designed to tackle musical problems usually focus on specific dimensions.

One of these problems consists of generating a melody given a chord sequence, which only involves note durations and pitches, there is no concept of polyphony or instrumentation, for example. The goal of this project is to design and implement a system able to generate a melody given a chord sequence, using information from a corpus to guide composition. Two of the main challenges of this kind of problems are the variables arrangement and the control of the global structure of the melody. Regarding variables, we made the assumption that a hierarchical organization would improve the system's flexibility which would make it easier to express constraints. For the structure, we designed an algorithm which uses suffix trees to detect repeating patterns in existing melodies and made the system able to replicate them in the result.
Our system is made of hierarchically organized blocs. The melody is made of bars which contain chords under which are located the notes. Each block has a variable number of notes which needs to be fixed first in order to instantiate the corresponding variables. This means that the system has to work in two phases. The first one assigns a rhythm pattern to every bar, which decides both the number of notes and their durations. The second phase fixes the pitch of every note of the melody.

To generate a melody, the user has to choose an element of the corpus and give its chord sequence to the system. For the first phase, he also has to give the list of the rhythm patterns of the chosen piece and their number of occurrences. To solve this part of the problem, the variables are fixed in ascending order of domain size and the values are assigned randomly following the distribution described by the frequencies of the rhythm patterns. The user has the option to use the frequencies from the original melody or from the whole corpus. Furthermore, he can add equality and cardinality constraints on the rhythm patterns, the

number of notes in each bar and on the note durations.

The second phase assigns a pitch to each note, from the first to the last. The resolution is guided by a COST-REGULAR constraint which automaton is built using statistics from the corpus. For each note, its position, the chord played at the same time and the previous note are taken into account and the statistics provide a distribution for the possible pitches given this context. Like in the first phase, the value selection heuristic uses the distribution. The user can once again impose equality and cardinality constraints on the pitches.

During both resolution phases, the system also respects the constraints coming from the structural patterns identified by the algorithm and specified by the user.

To evaluate the generated melodies, an online survey was made to rate the imitation capacity of the system as well as the general quality and diversity of the melodies. Nineteen people took the survey and five of them were musicians. The gathered results concerning these last two aspects are very good. The imitation part asked the participants to identify the original melody, which was composed by a human, among three melodies. Over forty percents of the answers were in favor of the system which means they identified a generated melody as the original. This data shows that it is not necessary to develop custom constraints nor explicitly add musical rules in order to generate credible and creative melodies using constraint programming.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

PPC	Programmation Par Contraintes
CSL	Computer Science Lab
CSP	Constraint Satisfaction Problem
IRCAM	Institut de Recherche et Coordination Acoustique/Musique
DDM	Diagramme de Décision Multivalué

LISTE DES ANNEXES

CHAPITRE 1 INTRODUCTION

Comment parler de musique sans mentionner l'informatique aujourd'hui ? Mis à part certaines performances en *live*, difficile de se passer d'un ordinateur dans la chaîne de création musicale. Auteurs, compositeurs, interprètes, producteurs, mais aussi professeurs font régulièrement usage des différents outils disponibles pour leurs activités respectives. L'évolution de l'intelligence artificielle a permis de développer de nouvelles méthodes de création, notamment de composition. Depuis les années 1950, la composition assistée par ordinateur profite des progrès de l'informatique pour accompagner les artistes dans leur travail. Parmi ces techniques on retrouve bien sûr l'apprentissage machine et bien d'autres méthodes [1, 2] mais aussi la programmation par contraintes. C'est cette dernière qui nous intéresse ici.

1.1 Programmation par contraintes

La Programmation Par Contraintes (PPC) est une forme de programmation déclarative qui est souvent utilisée pour aborder des problèmes combinatoires. Son efficacité provient de sa structure : en PPC, la description du problème et sa résolution sont clairement distinctes. La première partie est la responsabilité de l'utilisateur. La modélisation du problème se fait sous forme des variables qui sont mises en relation par des contraintes. Cette étape est extrêmement importante puisque c'est elle qui forme la représentation du problème et donc qui détermine les manipulations possibles. La résolution est ensuite effectuée par un solveur qui, en suivant une stratégie de recherche par défaut ou encore choisie par l'utilisateur, tente de trouver une solution qui respecte les contraintes mises en place.

1.1.1 Modélisation

Les choix de modélisation faits par l'utilisateur influencent directement le temps de résolution et la qualité des solutions trouvées. En effet, pour un même problème, il existe souvent plusieurs modèles possibles, structurés différemment, qui influent donc directement sur le processus de résolution. Pour modéliser un problème donné, on le représente généralement sous la forme d'un Problème de Satisfaction de Contraintes (en anglais Constraint Satisfaction Problem (CSP)). Un CSP est composé de trois éléments : un ensemble de variables X, un ensemble de domaines D associés à ces variables et un ensemble de contraintes C les mettant en relation.

Exemple 1.1 (Modélisation d'un sudoku)**.** Soit la grille de sudoku de la figure 1.1. Pour la

	2			3		9		7
	1							
4		7				2		8
		5	2				9	
			1	8		7		
	4				3			
				6			7	1
	7							
9		3		2		6		5

Figure 1.1 Exemple de sudoku

résoudre, il faut modéliser le problème sous la forme d'un CSP. Intuitivement, on va déclarer une variable par case de la grille, chacune avec le même domaine : les entiers de 1 à 9. Un premier ensemble de contraintes est alors imposé par l'état de départ de la grille car certaines valeurs sont déjà fixées. Puis, pour trouver une solution, il faut poser les contraintes tirées des règles du jeu : chacune des 9 lignes, chacune des colonnes et chacun des 9 carrés (délimité par les lignes plus épaisses) doivent contenir 9 chiffres différents. Cela se modélise parfaitement avec 27 contraintes ALLDIFFERENT. Cette dernière agit sur un ensemble de variables et fait en sorte qu'elles soient toutes fixées à une valeur différente. Si on identifie les lignes de a à i et les colonnes de 1 à 9, le CSP prends donc la forme suivante :

$$
\begin{aligned}
X &= \{xy \mid x \in \{a, b, c, \ldots, i\},\ y \in [1, 9]\} \\
D &= \{D_{xy} = [1, 9]\ \forall x \in [a, i],\ \forall y \in [1, 9]\} \\
C &= \{a2 = 2,\ a5 = 3,\ a7 = 9, \ldots,\ i7 = 6,\ i9 = 5, \\
&\quad \{AllDifferent(ligne_x) \mid x \in [a, i]\},\ \{AllDifferent(colonne_y) \mid y \in [1, 9]\}, \\
&\quad \{AllDifferent(carré_{xy}) \mid x \in \{abc, def, ghi\},\ y \in \{123, 456, 789\}\}\}
\end{aligned}
$$

Les variables d'un CSP peuvent être divisées en deux catégories. D'abord, les variables de branchement, qui constituent la structure principale du problème. Ce sont elles qui sont fixées par l'algorithme de résolution pour progresser vers une solution. Ensuite, il est possible d'introduire des variables auxiliaires pour faciliter l'expression de certaines contraintes. Les

valeurs des variables auxiliaires dépendent directement de celles des variables de branchement et ne sont pas importantes pour la solution finale. L'exemple du sudoku ne nécessite pas l'introduction de variables auxiliaires, les quatre-vingt-une variables du CSP sont des variables de branchement.

1.1.2 Résolution

La résolution d'un problème de satisfaction de contraintes consiste à affecter à chaque variable de branchement une valeur de son domaine en respectant les contraintes posées. Pour cela, le solveur emploie un algorithme spécifié par l'utilisateur ou par défaut. Il y a deux aspects importants à considérer dans cet algorithme : l'ordre de choix des variables et l'ordre de choix des valeurs. La combinaison d'une heuristique de choix de variable avec une heuristique de choix de valeur va donc former la stratégie de recherche du solveur. Pour profiter des avantages de la PPC, il est important de choisir une stratégie adaptée au CSP mis en place. La nature de la programmation par contraintes permet donc d'expérimenter avec plusieurs stratégies pour un même CSP ou avec différents CSPs en suivant une stratégie similaire.

À chaque choix de valeur fait par le solveur pour une variable donnée, la décision est propagée dans le réseau de contraintes. Ce processus de propagation interprète les contraintes impliquant la variable fixée et filtre des domaines des autres variables les valeurs incompatibles avec le choix effectué. Ces filtrages sont à leur tour propagés dans le réseau et en provoquent d'autres. Une solution est alors atteinte une fois que chaque variable est fixée à une valeur et que l'assignation respecte les contraintes posées. Dans le cas où un des choix a mené la recherche dans une impasse, la stratégie la plus simple consiste à revenir en arrière pour faire un autre choix. Il existe cependant de nombreuses autres façons de naviguer l'arbre de recherche et de gérer ce type de situation.

Exemple 1.2 (Résolution d'un sudoku)**.** Reprenons le sudoku de la figure 1.1. Considérons que l'heuristique de branchement choisie soit l'ordre lexicographique pour les variables et pour les valeurs. Le solveur va donc commencer par traiter la variable $a1$. Étant donné l'état actuel de la grille, en interprétant les contraintes ALLDIFFERENT posées, le solveur a déjà filtré le domaine de $a1$ à l'ensemble $\{5, 6, 8\}$. Il va donc assigner à $a1$ la valeur 5. Cette assignation va alors être propagée dans le réseau de contraintes et causer de nouveaux filtrages. Puisque $a1$ est impliquée dans 3 contraintes ALLDIFFERENT (ligne a, colonne 1, carré $abc123$), toutes les autres variables concernées par cette contrainte vont être affectées par la propagation et la valeur 5 sera filtrée de leurs domaines.

En suivant l'ordre lexicographique, le solveur va continuer avec la variable $a3$ qui a comme domaine $\{6, 8\}$ et va donc lui assigner la valeur 6. De la même façon, toutes les variables de

la ligne a, de la colonne 3 et du carré $abc123$ vont être affectées par ce choix et perdre la valeur 6 de leur domaine. Le processus continue de la sorte jusqu'à ce que toutes les variables aient été fixées.

Souvent, l'interprétation des contraintes d'un CSP donne lieu à des filtrages plus complexes. Dans notre exemple, la contrainte ALLDIFFERENT de la ligne i va forcer la variable $i6$ à prendre la valeur 1 puisque $i2$, $i4$ et $i8$ sont sujets à des contraintes qui ont filtré la valeur 1 de leurs domaines respectifs.

1.1.3 Les contraintes REGULAR, COST-REGULAR et TABLE

La contrainte COST-REGULAR [3] est la variante avec coûts de la contrainte globale REGULAR [4]. Cette dernière s'assure que les valeurs assignées à une séquence de variables $[X]$ de longueur n forment un mot reconnu par un automate fini déterministe A donné. On l'exprime alors simplement comme suit :

$$REGULAR([X], A)$$

La variante COST-REGULAR ajoute des coûts aux transitions de l'automate et allonge donc la liste des paramètres :

$$COST\text{-}REGULAR([X], A, z, C)$$

où z est la variable qui représente le coût total de l'assignation des variables de $[X]$, selon la matrice des coûts C. Il est alors possible d'optimiser le coût de la solution. On parle dans ce cas de problèmes d'optimisation plutôt que de problèmes de satisfaction.

TABLE [5] agit également sur une séquence de variables $[X]$ de longueur n mais permet cette fois-ci à son utilisateur de spécifier explicitement les séquences de valeurs autorisées. Ses paramètres sont donc le tableau de variables en question ainsi qu'un ensemble de n-uplets E :

$$TABLE([X], E)$$

Exemple 1.3 (Utilisation de REGULAR et TABLE)**.** Prenons l'exemple simple d'une séquence $[X]$ de trois variables X_1, X_2 et X_3 qui ont toutes les trois le même domaine $\{a, b, c\}$. Pour faire en sorte que les trois variables prennent chacune une valeur différente, il est possible de simplement poser une contrainte ALLDIFFERENT sur la séquence. Un comportement équivalent peut-être obtenu avec une contrainte REGULAR en utilisant l'automate de la figure 1.2.

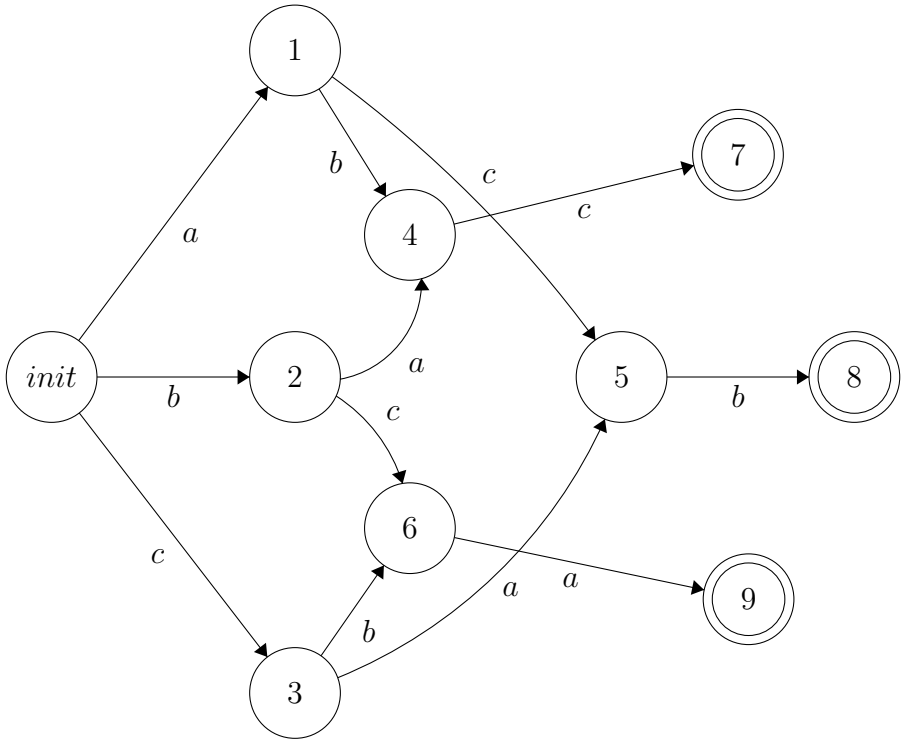


Figure 1.2 Automate pour contrainte REGULAR équivalente à un ALLDIFFERENT

Il est également possible de poser une contrainte TABLE et d'obtenir un modèle similaire. Il suffit de spécifier tous les triplets possibles : $TABLE([X], \{\{a,b,c\},\{a,c,b\},\{b,a,c\},\{b,c,a\},\{c,a,b\},\{c,b,a\}\})$.

1.2 PPC et musique

1.2.1 Intuition

Le parallèle entre la PPC et la musique se fait naturellement de par les caractéristiques du paradigme de programmation déclarative. En effet, comme l'explique T. Anders dans son excellente revue de littérature des systèmes de contraintes dédiés à la musique [6], la théorie de la musique est traditionnellement exprimée sous forme de règles. Celles-ci mettent en relation les différentes dimensions d'une pièce (hauteurs, durées, intervalles, relations entre voix...) dans le but de guider l'artiste dans la composition d'un morceau. Ainsi, pour atteindre son objectif, le compositeur va mettre en place les contraintes à respecter pour que le résultat de son travail s'inscrive dans le style de musique qu'il veut produire. Les règles de la musique se traduisent donc naturellement sous forme de contraintes. De plus, étant donnée la nature

mathématique de la musique, il est relativement facile de les traduire sous forme de code informatique.

1.2.2 Modélisation

Modéliser informatiquement un environnement musical est une tâche difficile. Lors de la composition, un artiste dispose d'un point de vue général de son travail et développe mentalement un réseau de relations entre tous les éléments de sa partition. Fournir à l'utilisateur d'un système informatique une vue d'ensemble de la même qualité est pratiquement impossible. Cependant, l'approche modulaire de la PPC permet de décomposer ce type de système complexe et donc de faire des compromis pour se concentrer sur les aspects importants du problème traité. Les nombreux problèmes du domaine de la musique (composition d'une mélodie, harmonisation, contrepoint, rythme...) peuvent ainsi tous être approchés différemment, en mettant en place un modèle adapté à l'objectif visé. Par exemple pour un problème d'harmonisation, on cherchera à développer un modèle qui favorise l'expression de contraintes sur les hauteurs des notes des différentes voix. Au contraire, un CSP destiné à décrire un problème rythmique sera construit de façon à pouvoir manipuler efficacement les variables liées à cette dimension. Les systèmes de contraintes dédiés à la musique (voir section 2.2) sont donc généralement conçus pour résoudre un type de problème en particulier, il est difficile de mettre au point un système totalement générique.

1.2.3 Exemple

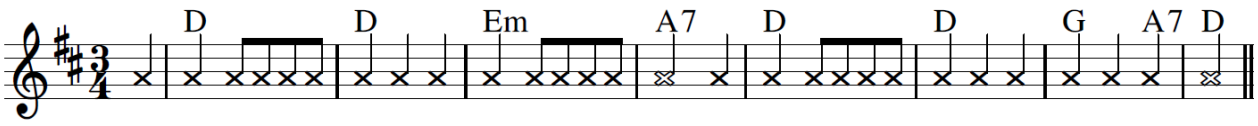


Figure 1.3 Accords et durées des notes des 9 premières mesures de la valse Caerdroea

Prenons l'exemple simple de génération d'une mélodie avec rythme et accords prédéfinis. La figure 1.3 nous donne la suite d'accords et les durées de notes à respecter. La seule dimension variable de notre CSP sera donc celle des hauteurs de notes. La structure rythmique imposée contient 28 notes, on instancie donc 28 variables, une pour chaque hauteur de note :

$$X = \{X_i \mid i \in [0, 27]\}$$

Notez que si nous voulions rendre la dimension rythmique variable, à moins d'avoir un nombre

de notes prédéfini (ce qui réduit beaucoup le nombre de possibilités), le nombre de variables de la dimension des hauteurs de notes serait lui-même une variable, ce qui augmenterait considérablement la complexité du problème.

Pour les domaines, nous avons plusieurs options possibles. Une première façon de fonctionner serait d'associer au do la valeur 0, puis d'incrémenter de 1 pour chaque demi-ton jusqu'au si. Nous pourrions aussi procéder de manière relative à la tonique du morceau, dans notre cas le ré puisque nous sommes en ré majeur. Ainsi, la valeur 0 correspondrait au ré et non pas au do. La représentation choisie influence directement l'expression des contraintes et donc la flexibilité du système. Prenons pour cet exemple la première option :

$$D = \{D_i = [0, 11] \ \forall i \in [0, 27]\}$$

Ce choix implique que les hauteurs des notes sont limitées à une seule octave, ce qui rend le problème plus simple mais implique une diversité de solutions plus faible. Il faudrait ajouter des paramètres auxiliaires ou bien changer de représentation pour pouvoir générer des notes qui s'étendent sur plusieurs octaves.

Les contraintes de notre CSP vont donc servir à s'assurer que la mélodie générée soit cohérente avec la suite d'accords imposée. Pour faire simple, il est nécessaire d'au minimum interdire aux variables de prendre des valeurs associées à des notes qui ne sont pas dans la gamme de ré majeur. Ensuite, il est possible d'ajouter d'autres contraintes pour guider la recherche, par exemple en imposant à la première note sous chaque accord de faire partie de ce dernier. Notre modèle nous permet aussi de poser des contraintes sur les intervalles, ce qui inclut, entre autres, l'interdiction des unissons consécutifs :

$$\begin{aligned} C = \{&X_i \in \{1, 2, 4, 6, 7, 9, 11\} \ \forall i \in [0, 27], \\ &X_i \in \{2, 6, 9\} \ \forall i \in \{1, 6, 16, 21, 27\}, \\ &X_9 \in \{4, 7, 11\}, \\ &X_i \in \{1, 4, 7, 9\} \ \forall i \in \{14, 26\}, \\ &X_{24} \in \{2, 7, 11\}, \\ &Si \ X_{i-1} = X_i \ alors \ X_{i+1} \neq X_i \ \forall i \in [0, 27]\} \end{aligned}$$

La stratégie de recherche à adopter pour la résolution dépend fortement du modèle mis en place. Dans cet exemple, les choix de variables pourraient se faire par ordre croissant de taille de domaine (heuristique $minSize$) pour pouvoir fixer d'abord les premières notes sous chaque accord. En ce qui concerne les choix de valeurs, une heuristique aléatoire serait suffisante dans

un cas aussi simple.

Prenez note que le CSP mis en place dans cet exemple sert simplement à illustrer la modélisation d'un problème musical avec la PPC. Les variables, domaines et contraintes détaillés ici ne sont donc pas représentatifs du fonctionnement du système décrit à la section 3.

1.3 Objectifs du projet

L'objectif principal de ce projet est de concevoir et développer un système de contraintes permettant de générer une mélodie de façon automatique en utilisant des informations tirées de pièces existantes. Ces informations peuvent être interprétées pour imiter les propriétés d'un morceau, mais également pour inspirer des créations originales. Ainsi, de notre objectif principal se dérivent deux sous objectifs : l'imitation d'une mélodie existante en reprenant sa suite d'accords et sa structure globale ; et la création d'une mélodie originale en donnant plus de flexibilité à l'utilisateur.

La crédibilité d'une mélodie repose évidemment sur sa suite de notes qui doit être musicalement cohérente par rapport à la suite d'accords qui l'accompagne. Le système devra donc être capable d'utiliser les mécanismes de la PPC pour exprimer ces relations musicales qui font en sorte que le résultat est agréable à écouter. Un second aspect à ne pas négliger est la structure de la pièce générée. Souvent, une mélodie contient un thème qui est répété régulièrement et d'autres motifs (rythmiques ou mélodiques) plus courts qui reviennent également. C'est un type d'information qu'il est possible d'extraire des mélodies existantes.

Pour atteindre ces objectifs, le système devra être modélisé de façon à pouvoir manipuler facilement les caractéristiques principales de la mélodie, c'est-à-dire les hauteurs et les durées des notes, qui constituent donc nos dimensions variables. Puisque le système se limite à la génération d'une seule voix, les difficultés liées à la représentation de la polyphonie ne sont pas à prendre en compte. De plus, aucune notion d'instrumentation n'est énoncée dans les objectifs, ce qui réduit aussi les dimensions du problème.

CHAPITRE 2 REVUE DE LITTÉRATURE

La composition assistée par ordinateur est un domaine qui suscite l'intérêt des chercheurs depuis les débuts de l'informatique. La littérature sur le sujet est dense mais aussi très variée étant donné les nombreuses approches possibles et les différents problèmes qui constituent ce secteur de recherche. La programmation par contraintes n'est qu'une méthode parmi tant d'autres pour appréhender des défis musicaux de manière informatique, notamment la génération automatique de mélodie.

2.1 Historique

La référence historique en matière de composition assistée par ordinateur date de 1957 et s'intitule l'*Illiac Suite* [7]. Il s'agit du résultat de quatre expériences de composition pour un quatuor à cordes avec l'ordinateur Illiac à l'Université d'Illinois. Chacune des quatre expériences correspond à un des quatre mouvements de la pièce générée. Bien qu'à l'époque, la PPC n'existait pas comme on la connait aujourd'hui, on retrouve tout de même la notion de règles et de contraintes dans la technique utilisée. Ainsi, la première expérience introduit uniquement une sélection limitée de règles de contrepoint puis la seconde en ajoute d'autres. L'expérience 3 s'intitule *Experimental Music* et ajoute des contraintes basiques sur le rythme et les hauteurs de note à utiliser. Finalement, dans la quatrième partie, les auteurs expérimentent avec les chaînes de Markov dans un contexte musical.

En 1980, Kemal Ebcioglu développe un programme capable de générer une mélodie qui respecte les règles du contrepoint rigoureux selon un *Cantus Firmus* (mélodie servant de base au contrepoint) [8]. Pour que le résultat soit satisfaisant, il ajoute même ses propres contraintes. Avec le *Cantus Firmus* et l'ensemble des règles à respecter en entrée, l'algorithme cherche selon une stratégie très simple et énumère toutes les solutions qu'il trouve. Un effort de modélisation est fait pour être capable de définir des règles complexes sous forme de code. En 1984, Bill Schottstaedt poursuit la recherche en traitant ce même problème mais étendu jusqu'à six voix différentes [9]. Pour se rapprocher encore plus de la théorie qui dit que les règles de la musique sont des recommandations plutôt que des directives, il introduit des pénalités pour chaque contrainte, selon leur importance et cherche alors à minimiser la pénalité totale du résultat. Quelques années plus tard, en 1990, K. Ebcioglu présente son système appelé CHORAL [10]. Beaucoup plus complet et complexe, tant au niveau des règles que de la stratégie de recherche, ce système est capable d'harmoniser des chorales à 4 voix dans le style de J.S. Bach. Le système inclut environ 350 règles issues de l'analyse des chorales

de J.S. Bach et qui guident l'harmonisation et la génération de mélodies. En ce qui concerne la modélisation du problème, K. Ebcioglu a dû créer son propre langage de programmation logique (BSL) pour arriver à ses fins, ce qui démontre une évolution considérable.

Puis, en 1996, Russell Ovans et Rod Davison proposent un modèle de PPC sous forme de CSP et insistent donc sur le potentiel d'une telle approche qui permet de séparer clairement la stratégie de recherche de la définition des contraintes [11]. Leur système se présente sous forme d'une interface graphique qui donne à l'utilisateur la responsabilité de composer une mélodie respectant les règles de contrepoint selon une mélodie générée automatiquement par le système. Ce dernier s'assure simplement que l'utilisateur ne viole aucune de ces règles en affichant seulement les notes qui les respectent.

Ce projet se concentre sur la génération d'une mélodie, un problème relativement simple à définir comparé à d'autres dans le domaine. La plupart des travaux qui concernent la composition automatique reposent sur un modèle de Markov pour guider la génération. Ces modèles permettent de mettre en place des techniques d'apprentissage pour s'inspirer de pièces existantes et imposer une certaine structure à court terme. Davismoon et Eccles proposent de combiner une chaîne de Markov avec des contraintes musicales exprimées de diverses façons [12]. Ils contrôlent ainsi la forme de la mélodie avec des techniques d'optimisation stochastique et mettent en place un recuit simulé pour guider la recherche de solution. Les modèles de Markov ont également été jumelés avec la PPC, notamment dans les travaux de F. Pachet et son équipe du Sony Computer Science Lab (CSL) à Paris. Ces derniers ont mis au point ce qu'ils appellent des contraintes de Markov [13]. En reformulant un modèle de Markov sous forme de CSP, ils peuvent alors utiliser les techniques de propagation propres à la PPC et ajouter des contraintes additionnelles pour décrire précisément le résultat désiré [14]. L'équipe étend ensuite sa technique à la génération de *lead sheets* au complet, c'est-à-dire une mélodie avec une suite d'accords [15].

D'autres chercheurs optent pour des méthodes différentes. C'est le cas de G. Boenn et al. qui critiquent les processus de Markov pour leur aspect probabiliste trop prévisible, leur forme de chaîne qui force une progression du début à la fin et leur manque de contrôle global de la mélodie [16]. En utilisant un autre type de programmation déclarative (l'*answer set programming*), ils mettent au point un système, ANTON, capable de générer des mélodies et de trouver des erreurs dans des compositions. Les auteurs espèrent poursuivre leur travail et faire en sorte que la qualité des résultats du système soit analysée par des humains et que les informations récoltées soient redonnées au système pour qu'il infère des nouvelles règles pour améliorer la génération.

Certains travaux se concentrent davantage sur le processus de composition. Dans [17], les auteurs ajoutent des niveaux d'abstraction pour modéliser l'intention du compositeur et ainsi contrôler quelles contraintes sont posées selon ce qu'il veut transmettre à travers sa musique.

En ce qui concerne les autres types de problèmes du domaine musical et les autres techniques utilisées pour tenter des les résoudre, les revues de littérature [1] et [2] constituent une excellente source d'informations.

2.2 Systèmes de contraintes musicaux

De nombreux systèmes dédiés à la résolution de CSP musicaux ont été développés au fil des années. Certains sont conçus pour résoudre un problème précis alors que d'autres fournissent un environnement de travail générique. Cette section dresse une courte liste des plus populaires.

OMClouds [18] est un système dédié à la programmation par contraintes appliquée à la musique et il est implémenté en Common Lisp. Il est basé sur le système OpenMusic, un environnement visuel de composition assistée par ordinateur développé à l'Institut de Recherche et Coordination Acoustique/Musique (IRCAM).

OMClouds est facile d'utilisation grâce à sa représentation simple sous forme de listes de variables et son interface graphique. Cependant, cette simplicité implique des fonctionnalités limitées donc le système n'est pas adapté à n'importe quel CSP musical. Pour résoudre un CSP, OMClouds fonctionne uniquement par recherche heuristique en traduisant les contraintes strictes en contraintes souples et fonctions de coût. Les contraintes souples indiquent une préférence plutôt qu'une nécessité et rendent alors le CSP plus flexible. Les fonctions de coût fournissent un objectif d'optimisation pour se rapprocher le plus possible de la solution qui aurait été obtenue en gardant des contraintes strictes. Ces manipulations permettent donc de réduire la complexité de la définition de certains CSP et ainsi de faciliter leur traitement.

PWGLConstraints [19] est une librairie du langage PWGL (PatchWorkGL), un langage de programmation visuelle dédié à la composition par ordinateur et implémenté en Common Lisp. La librairie PWGLConstraints ajoute à ce langage deux composantes : PMC, un langage général de programmation par contraintes pour des types de données simples, et Score-PMC, un langage spécialement conçu pour traiter les problèmes polyphoniques.

Dans PWGLConstraints, les variables sont de simples listes d'entiers qui peuvent représenter

les hauteurs des notes (format MIDI [20]) ou leurs durées par exemple. Dans le cas particulier de Score-PMC, la structure rythmique du morceau doit être définie en avance et les relations entre les dimensions musicales sont préétablies. Pour exprimer ces contraintes, le système utilise des techniques de filtrage par motif (*pattern matching*) ce qui permet à l'utilisateur de définir ses propres contraintes. Le système supporte aussi des contraintes heuristiques. Pour la recherche de solutions, un simple algorithme de *backtracking* chronologique est utilisé et ses performances peuvent être améliorées en utilisant en plus le *forward checking*.

Les créateurs de PWGLConstraints présentent également le système Situation [19], qui a tout d'abord été développé en tant que librairie du langage PatchWork avant d'être porté sur OpenMusic. À l'origine, Situation a été écrit en Common Lisp par Camilo Rueda en collaboration avec le compositeur Antoine Bonnet dans le but de résoudre des CSP harmoniques.

La musique est représentée sous forme de séquence d'objets. Ces derniers sont composés de points et Situation définit des distances entre ces points (au sein d'un même objet mais aussi entre deux objets différents) et l'interprétation est laissée à l'utilisateur. Par exemple, les points peuvent représenter les hauteurs des notes et les objets des accords. Les distances permettent alors de mettre les points et les objets en relation. Des contraintes adaptées à la composition sont prédéfinies et des mécanismes sont disponibles pour permettre à l'utilisateur de définir ses propres contraintes ainsi que leur portée. Le système supporte également des contraintes souples. La recherche de solutions se fait à l'aide de plusieurs variations de *forward checking*, mécanisme qui consiste à vérifier les répercussions d'un choix de valeur sur les autres variables du problème pour évaluer ou choisir une certaine assignation.

MusES est un autre système né dans les années 90 [21]. Il s'agit d'un environnement permettant de travailler sur des problèmes musicaux dans un contexte orienté objet. En combinaison avec la librairie BackTalk [22] du langage SmallTalk, le système représente un excellent outil pour résoudre des CSP musicaux.

La représentation de la musique dans MusES se fait sous forme de collections composées d'objets temporels. Ces objets possèdent des informations sur leur temps de début ainsi que leur durée. Les collections s'assurent que leurs objets sont rangés selon leur temps de début. Pour définir des contraintes sur ces variables, il faut utiliser directement le langage BackTalk ce qui rend le travail intuitif mais très limité. En effet, il est très difficile d'exprimer des contraintes complexes sur de nombreuses variables. BackTalk propose une librairie d'algorithmes de recherche et l'utilisateur peut ainsi choisir le plus efficace selon son problème.

Plus récemment, le système Strasheela [23] a été développé par Torsten Anders dans le but de combiner les avantages des systèmes décrits précédemment. Il est implémenté avec le langage Oz et tente de trouver un équilibre entre généralité et efficacité pour être utile pour un maximum de CSP musicaux.

Dans Strasheela, l'utilisateur manipule des blocs qu'il peut aussi créer lui-même en mettant à profit les relations d'héritage du paradigme de programmation orientée objet. Ces blocs sont rangés dans des conteneurs et l'information est accessible dans n'importe quelle direction : les objets ont accès aux informations des conteneurs et inversement. Cela permet de mettre en place une structure hiérarchique et de favoriser la propagation de contraintes. Les contraintes se présentent sous forme de fonctions et l'utilisateur les manipule à haut niveau ce qui rend le tout plus intuitif. Strasheela utilise les techniques de recherche du langage Oz et permet ainsi à l'utilisateur de programmer lui-même son propre algorithme de recherche.

2.3 Travaux du Sony Computer Science Lab à Paris

Les travaux de François Pachet et son équipe au Sony CSL Paris représentent la source d'inspiration principale de ce projet. C'est en étudiant leurs articles et en découvrant leur système *FlowComposer* que nous avons débuté.

FlowComposer [24] est une application web qui aide son utilisateur à composer de la musique sous forme de *lead sheets* (partition simplifiée sur laquelle figurent la mélodie et les accords).

Grâce à leur travail sur les chaînes de Markov et à leur contrainte globale METER [25], les auteurs proposent un système basé sur la programmation par contraintes capable de composer de la musique en imitant le style d'un corpus sélectionné par l'utilisateur. Ce dernier est également capable de décider s'il veut composer un morceau au complet ou simplement compléter une section d'une *lead sheet* existante. FlowComposer fait actuellement partie d'un ensemble d'outils basés sur le web qui sont utilisés dans des productions musicales professionnelles.

La contrainte METER [25] a été définie par Pierre Roy et François Pachet en 2013. Son objectif est de faire en sorte qu'une séquence de notes soit valable d'un point de vue métrique. Plus précisément, METER impose à une séquence de notes : qu'elle respecte les propriétés d'une chaîne de Markov donnée, qu'elle soit d'une durée spécifique et qu'aucune de ses notes ne traverse une barre de mesure.

Pour utiliser cette contrainte il faut tout d'abord construire le modèle de Markov qui in-

dique les probabilités de transition entre les différentes notes possibles. Pour cela, un corpus composé de plusieurs morceaux sélectionnés en fonction de l'objectif désiré est analysé pour apprendre toutes les transitions possibles et leurs fréquences d'apparition. De cette façon, en formant le corpus avec un ensemble de pièces d'un même auteur par exemple, on peut imiter son style de composition.

METER peut être exprimée sous la forme d'un automate en utilisant la contrainte REGULAR. Les états de cet automate représentent alors un point dans le temps et les transitions sont étiquetées par les notes constituant la chaîne de Markov. Pour échantillonner cet automate, les chercheurs du Sony CSL Paris l'expriment sous la forme d'une distribution de probabilités sur les séquences possibles, en fonction des transitions de la chaîne de Markov. Ensuite, ils la transforment en une nouvelle distribution en considérant les couples $(note, état\ de\ l'automate)$ plutôt que les notes seules, ce qui leur permet d'obtenir un graphe de facteurs en arbre et contenant uniquement des facteurs binaires. Ils sont alors capables d'appliquer la technique de *belief propagation*. Cette dernière met en place des échanges de messages entre les différents nœuds du graphe et permet de calculer la distribution marginale de chacun de ces nœuds. De cette façon, les auteurs sont capables d'échantillonner le graphe de façon exacte en temps quadratique (par rapport à la durée de la séquence à générer) pour générer des séquences musicales satisfaisant la contrainte METER et imitant le style du corpus de départ [26].

Les chercheurs du Sony CSL Paris identifient dans le cadre de leur recherche le besoin d'une contrainte globale permettant de décrire la structure d'une séquence temporelle. Pour répondre à ce besoin, ils proposent la contrainte globale ALLEN [27]. Généralement, une séquence temporelle est modélisée par une liste de variables qui représentent les événements ayant lieu dans cette période de temps. Cependant, cette représentation pose problème lorsque l'on cherche à poser des contraintes sur les variables de cette liste puisque leurs positions temporelles dépendent des durées des événements précédents et non pas de leur indice dans la séquence. Il est donc difficile d'accéder directement aux variables ciblées. La contrainte ALLEN tente de résoudre ce problème en utilisant l'algèbre des intervalles de James F. Allen. Implémentée (dans sa version la plus efficace) avec des Diagrammes de Décision Multivalués (DDMs), ALLEN permet de mettre en relation les positions temporelles d'un ensemble de variables avec leurs indices. Dans le cas d'une séquence de notes, cela permet donc de poser des contraintes d'égalité entre les notes de deux mesures différentes par exemple. ALLEN est décrite par ses auteurs comme une généralisation de METER, évoquée plus tôt.

Pour faire simple, la contrainte s'exprime de la manière suivante :

$$ALLEN_{[a,b]}([X], \varepsilon, I)$$

avec a et b les temps de début et de fin de l'intervalle visé, $[X]$ le tableau de variables représentant la suite de notes (une variable par note), ε l'ensemble des événements survenant entre a et b (événement = couple $(hauteur, durée)$) et I l'ensemble des indices des variables de $[X]$ entre a et b. En posant des contraintes additionnelles sur ε et sur les variables spécifiées par I, il est donc possible de limiter les événements possibles dans un certain intervalle de temps ou encore de mettre en place des égalités entre séquences de variables.

Les auteurs fournissent un exemple de l'utilisation de ALLEN dans un contexte musical. Pour une mélodie de durée d et une durée de note minimum de l, ils définissent dans un premier temps un tableau de d/l variables. Celles-ci représentent les notes de la mélodie et leurs domaines sont composés de couples $(hauteur, durée)$ comme décrit plus tôt. Les auteurs posent alors une contrainte ALLEN différente pour chaque accord (en les identifiant par l'intervalle de temps dans lequel ils apparaissent) afin de spécifier l'ensemble ε des événements possibles sous chacun d'entre eux. Ensuite, pour exprimer une égalité entre deux portions de mélodie avec la contrainte ALLEN, il faut tout d'abord poser deux de ces contraintes, une sur chaque portion concernée, identifiée par leurs positions temporelles de début et de fin. La contrainte ALLEN permet alors d'accéder aux indices des variables (ensemble I) représentant les notes des portions et donc de poser des contraintes d'égalité sur les séquences de notes, séquences qui d'ailleurs sont de longueur variable et requièrent donc une contrainte dédiée pour ce cas d'utilisation (appelée SEQEQ).

Les résultats obtenus sont convenables, cependant, la modélisation du problème n'est pas très intuitive. Définir dès le départ toute la liste de notes signifie qu'à moins qu'elles ne soient toutes de durée minimum, une partie des variables définies ne seront pas utilisées. Cela ne semble pas être la manière la plus efficace de procéder. De plus, cela demande un effort supplémentaire pour "aligner" les notes sous les accords et autres portions auxquelles l'utilisateur souhaiterait accéder. Il serait plus naturel de diviser la partition en blocs, organisés de manière hiérarchique, un peu à la façon de T. Anders dans son système *Strasheela* [23]. Cela donnerait plusieurs niveaux d'accès aux variables et donc plus de flexibilité pour définir des contraintes entre les notes, ce qui résoudrait en partie les problèmes identifiés par les chercheurs du Sony CSL Paris. En utilisant une organisation hiérarchique, l'accès aux variables contenues dans une certaine mesure serait instantané de par la structure du système, ce qui allégerait donc la résolution et rendrait l'utilisation plus intuitive.

Exemple 2.1. La figure 2.1 montre un exemple de modélisation d'une contrainte d'égalité entre deux mesures exprimée avec des contraintes ALLEN sur la gauche puis avec une simple contrainte d'égalité sur la droite grâce à une organisation hiérarchique. Dans ce dernier cas,

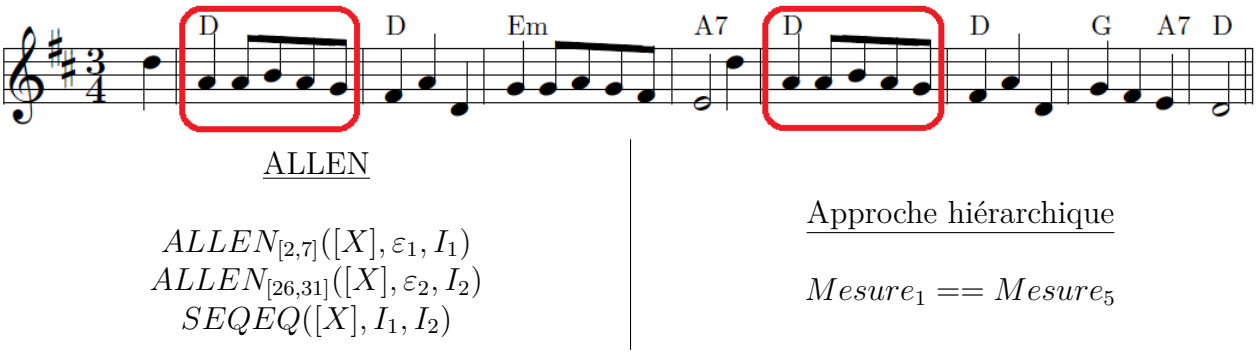


Figure 2.1 Exemple d'utilisation de la contrainte ALLEN et comportement équivalent avec une organisation hiérarchique.

puisque les nombres de notes de chaque mesure sont définis dans une phase antérieure, les sous-séquences concernées sont déjà de la même longueur. Pour la contrainte ALLEN, l'unité de temps utilisée est la croche et la première croche correspond au temps zéro.

2.4 Méthodes d'identification de patrons

Comme recensé dans la revue de littérature [28], il existe plusieurs méthodes relativement différentes pour identifier des patrons récurrents dans une pièce musicale. Une grande majorité de ces méthodes sont basées sur la représentation des différentes dimensions de la musique sous forme de chaînes de caractères. Ainsi, des algorithmes d'analyse provenant notamment de la bio-informatique peuvent être utilisés.

Parmi ces approches, certaines se présentent sous forme d'algorithmes qui traitent directement la chaîne de caractères en tant que telle, c'est le cas dans [29] et [30] notamment. D'autres mettent en place des structures de données comme des graphes ou des arbres permettant d'utiliser d'autres types d'algorithmes. En particulier, les arbres et tableaux des suffixes sont régulièrement utilisés pour analyser des chaînes de caractères, y compris dans le domaine de la musique.

Un arbre des suffixes est une structure de données qui a été introduite pour la première fois en 1973 pour des applications de filtrage par motif [31]. Il s'agit donc d'un arbre qui contient tous les suffixes d'une chaîne de caractères. Une branche est étiquetée par une sous-chaîne et mène à un nœud dont les branches représentent tous les suffixes de ladite sous-chaîne. Chaque nœud possède donc au moins deux enfants. En comptant le nombre de feuilles dans le sous-arbre d'un nœud, on peut donc savoir combien de fois est répétée la sous-chaîne qu'il décrit. Lorsqu'on parcourt l'arbre de la racine jusqu'à une feuille, on décrit un suffixe de la

chaîne de caractères, identifié par son indice de départ. Un arbre des suffixes possède donc autant de feuilles que de caractères. Pour s'assurer qu'aucun suffixe ne soit le préfixe d'un autre, un caractère de fin de chaîne est ajouté. La figure 2.2 montre un exemple d'arbre des suffixes.

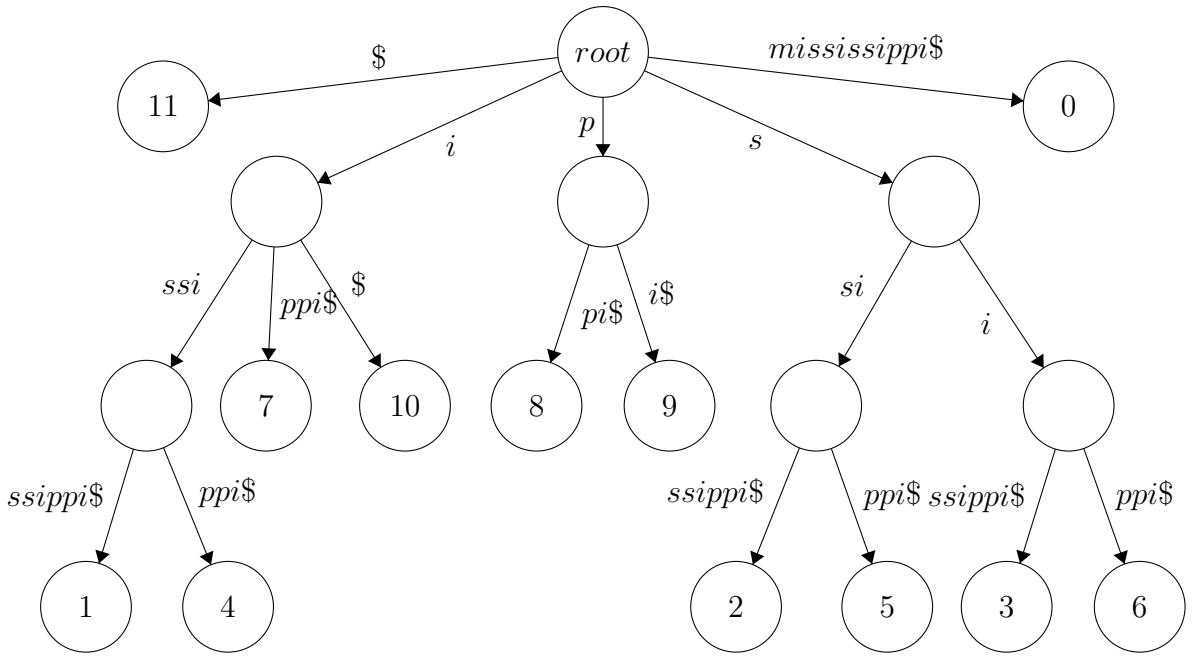


Figure 2.2 Exemple d'arbre des suffixes pour la chaîne de caractères *mississippi*

Ces structures de données sont notamment utilisées pour résoudre des problèmes comme celui de la plus longue sous-chaîne commune ou celui de la plus longue sous-chaîne répétée en temps linéaire dans le nombre de caractères.

Ainsi, Jekovec et al. utilisent ces structures de données pour l'analyse de mélodies et l'identification des patrons principaux [32]. L'utilisation des arbres des suffixes est fortement liée à la représentation des données. Dans leurs travaux, les auteurs mettent au point des représentations qui combinent hauteurs, intervalles et durées, ce qui leur permet de ne construire qu'un seul arbre des suffixes pour toute une mélodie. Leur travail est intégré dans leur logiciel d'analyse musicale *Harmonia*. Une autre façon de procéder consiste à choisir de se concentrer sur une certaine dimension de la mélodie et de construire la structure de donnée et l'algorithme de parcours en conséquence [33]. Il est également possible d'utiliser des séquences dérivées de la suite de notes principale telles que des inversions ou des rétrogrades et de mettre en place un tableau des suffixes (autre représentation d'un arbre des suffixes qui présente des avantages en termes de consommation de mémoire) pour chaque séquence [34].

Finalement, certains travaux vont plus loin et développent leurs propres versions des arbres de suffixes pour répondre à des besoins en particulier, notamment en termes de performance et de consommation de mémoire. C'est le cas des arbres de suffixes jumeaux de W. Lee et A. Chen par exemple [35]. Un exemple d'arbre des suffixes pour une courte mélodie est présenté plus loin, à la section 3.1.4.

D'autres chercheurs favorisent des méthodes moins explicites telles que des algorithmes matriciels [36] ou de forage de données (*Data Mining*) [37], ou encore des approches géométriques [38].

2.5 Méthodes d'évaluation des résultats

Le caractère subjectif de la musique fait de l'évaluation des résultats d'un système de composition automatique un défi à part entière. Il n'existe pas de méthode exacte pour qualifier la qualité d'une pièce et différents objectifs impliquent différentes méthodes d'évaluation. En effet, on cherche généralement à générer des pièces musicalement cohérentes mais significativement différentes de morceaux existants pour éviter toute forme de répétition ou de plagiat. Ainsi, des méthodes d'évaluation comme des fonctions objectif ou la distance de Hamming par exemple ne sont pas vraiment adaptées à ce domaine.

L'évaluation des compositions informatiques a été peu explorée et n'a pas suivi le rapide progrès des techniques mises au point par les chercheurs. En 2001, M. Pearce et G. Wiggins proposent finalement un cadre d'évaluation pour ce genre de travail [39]. Ils indiquent dans un premier temps que les objectifs des créateurs du système doivent être clairement définis pour que l'évaluation soit faite selon les bons critères. Puis, ils mentionnent que souvent, une première évaluation est faite par le système lui-même lors de la génération. En effet, dans le cas de la PPC, l'introduction de contraintes souples et d'un objectif d'optimisation constitue déjà une forme d'évaluation. L'élément principal de leur proposition est en fait une sorte de test de Turing, durant lequel des sujets doivent évaluer les résultats produits par la machine en essayant de les distinguer de pièces composées par un humain. Ainsi, selon les critères visés par les créateurs du système, on voudra peut-être se concentrer plutôt sur la structure rythmique du morceau évalué ou encore la ressemblance avec un certain style visé. Ils proposent aussi d'évaluer le degré de variation entre les résultats de plusieurs exécutions d'un même système. Cette approche est utilisée telle quelle par M. Shan et S. Chiu pour évaluer leur système basé sur la découverte de patrons [36]. Ils ajoutent également une évaluation des performances de leur système en termes de temps de calcul.

Certains autres chercheurs ont fait évoluer cette méthode en introduisant la possibilité pour le sujet de quantifier (sur une échelle de 1 à 5 par exemple) la ressemblance stylistique entre une pièce originale et une pièce générée et de laisser des commentaires expliquant ses choix [40]. D'autres vont encore plus loin en utilisant différentes versions d'un même système, en sélectionnant leurs sujets selon leur niveau de connaissance de la musique et en appliquant des méthodes d'analyse statistique sur les résultats obtenus par plusieurs sujets [41].

CHAPITRE 3 MISE EN ŒUVRE

3.1 Architecture du système

3.1.1 Objectifs et approche

Comme mentionné à la section 1.3, l'objectif de ce projet est de mettre au point un système capable de générer une mélodie crédible à partir d'informations recueillies automatiquement et avec des méthodes simples, en spécifiant un minimum de règles musicales explicites dans le but de conserver une certaine liberté.

L'approche développée utilise des informations rythmiques et d'enchaînement de notes extraites d'un corpus de mélodies et la mélodie générée doit respecter la structure et la suite d'accords d'une mélodie donnée. Dans un premier temps, le résultat obtenu sera fortement inspiré de la mélodie d'origine choisie puis, par la suite, plus de contrôle sera donné à l'utilisateur qui pourra alors générer des résultats plus variés. Les seules variables du problème sont donc les durées des notes (et par conséquent le nombre de notes de la mélodie) ainsi que leurs hauteurs.

La revue de littérature de la section 2 a fait ressortir deux besoins principaux. Tout d'abord, la structure des variables doit favoriser l'expression des contraintes et l'architecture du système doit donc permettre d'accéder aux variables de notre choix facilement. Pour répondre à cela, nous avons décidé d'adopter une structure hiérarchique, organisée en blocs. Chaque bloc représente une séquence de notes de longueur variable : l'ensemble des notes sous un certain accord, les notes dans une certaine mesure ou une portion arbitraire de la mélodie. Cette dernière peut donc être vue comme une séquence de blocs, peu importe par quel niveau on y accède. Tous les blocs peuvent ainsi avoir leurs propres contraintes, que ce soit sur la longueur de leur séquence ou sur les notes elles-mêmes. Cette structure rend également naturelle l'expression de contraintes entre les blocs telles que des égalités ou des transpositions. L'organisation choisie permet aussi au système d'être relativement générique en termes de style de musique et du type de contraintes possibles. Tous les détails de la structure du système sont donnés à la section 3.1.2.

Le second aspect mis en avant par la revue de littérature concerne la structure de la mélodie. En effet, certains travaux notent le manque d'organisation de leurs résultats qui ne présentent pas vraiment de phrases répétitives comme on le retrouve habituellement dans la musique. Une des fonctions du corpus mentionné précédemment est de répondre à ce besoin. En analysant la mélodie du corpus servant d'inspiration pour une exécution du système, on peut faire

ressortir les patrons répétitifs et imposer des contraintes permettant de les répliquer dans le résultat. La structure hiérarchique expliquée plus tôt favorise l'expression de ces contraintes puisqu'un patron peut-être représenté par un bloc du système. L'algorithme utilisé pour la détection des phrases répétitives fait intervenir des arbres des suffixes (*suffix tree* en anglais) et est détaillé à la section 3.1.4.

3.1.2 Structure des variables

Notre système est donc formé de blocs qui représentent différentes séquences de notes, il en existe cinq types :

— les blocs de notes forment la plus petite unité de la structure, ils représentent une petite fraction de la mélodie ;

— les blocs d'accords sont composés de blocs de notes et regroupent les notes qui se trouvent sous un accord. Il y en a donc un par accord ;

— les blocs de mesures représentent une mesure de la mélodie et contiennent les blocs d'accords inclus dans cette mesure ;

— la ligne mélodique est un bloc unique qui regroupe tous les blocs de mesure et donc toutes les notes de la mélodie ;

— les blocs de patrons représentent les occurrences des patrons structurels et sont constitués de pointeurs sur des blocs de notes.

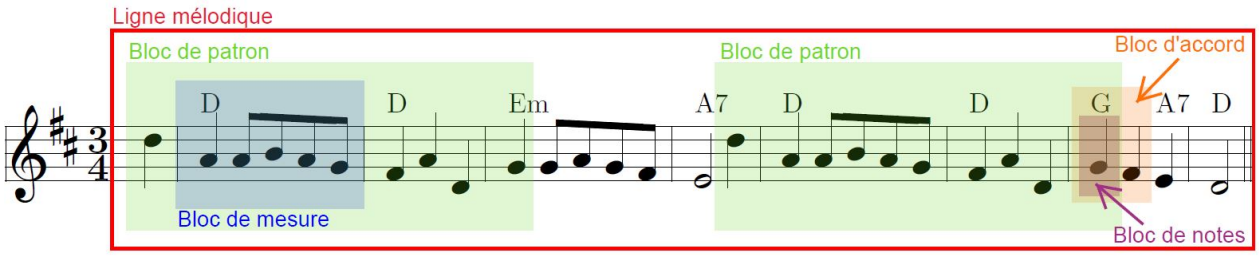


Figure 3.1 Structure hiérarchique du système

Comme on peut le voir sur la figure 3.1, l'intérêt des blocs de notes est de pouvoir créer des blocs de durée arbitraire, qui ne sont pas forcément alignés sur des blocs d'accords. C'est souvent le cas pour les blocs de patrons notamment. Autre point important : les accords ne traversent jamais une barre de mesure. Si la suite d'accords choisie contient un élément plus long qu'une mesure, l'utilisateur doit le diviser dans le fichier d'entrée qu'il fournit au système.

Chaque bloc possède donc un nombre de notes variable qu'on qualifiera de longueur du bloc, par opposition à sa durée qui représente la somme des durées des notes qui le composent. Cette particularité a un impact capital sur le fonctionnement du système puisqu'elle implique que le nombre de variables dans chaque bloc (hauteurs et durées des notes) est lui-même variable. Étant donné que le solveur utilisé (IBM ILOG Solver) ne propose pas de structure de données permettant de stocker un nombre variable de variables, il est nécessaire de fonctionner en deux phases de résolution distinctes : une première phase pour fixer le nombre de notes de chaque bloc de notes (et par conséquent de tous les blocs du système) afin de pouvoir instancier les tableaux qui contiendront les variables de hauteur et de durée qui seront fixées dans la seconde phase. Ces deux étapes sont décrites aux sections 3.2 et 3.3.

D'un point de vue technique, chaque type de bloc est implémenté par une classe dérivée de la classe abstraite de base `Block`. Le digramme de classes de la structure est présenté à la figure 3.2

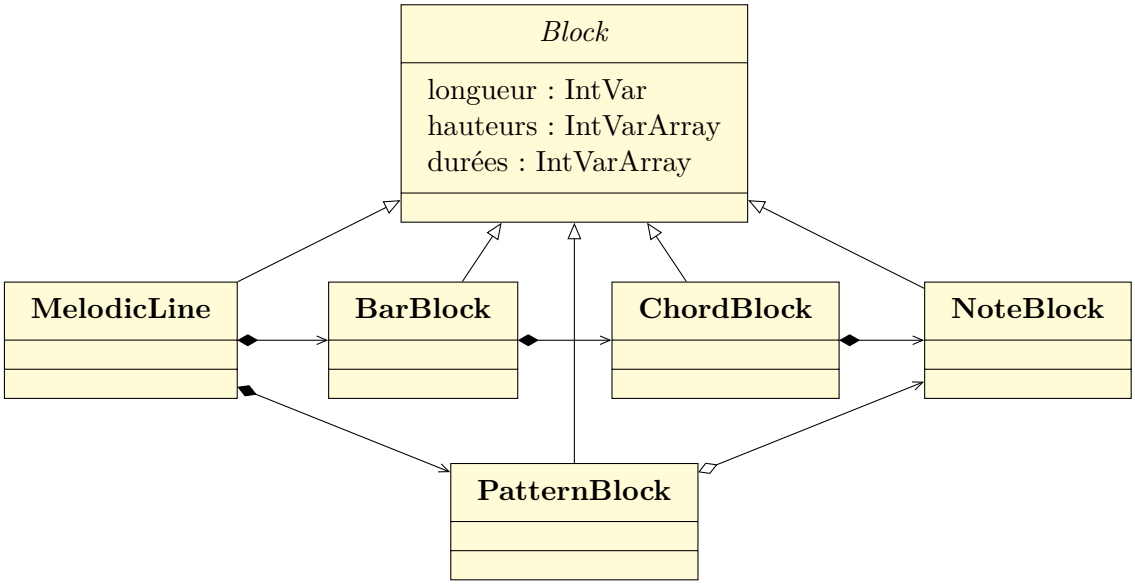


Figure 3.2 Diagramme de classes de la structure en blocs

3.1.3 Description du corpus

Pour avoir accès à des pièces de référence et à des statistiques pour guider la composition, le système a recours à un corpus d'environ une centaine de morceaux. Ce dernier est tiré de la base de données Nottingham [42], un ensemble de 1200 chansons folk anglaises et américaines disponible aux formats abc [43] et MIDI.

De ces 1200 chansons, nous avons décidé de nous concentrer sur un style en particulier et de ne garder que les valses. Deux raisons principales ont motivé ce choix : premièrement, les mélodies de valses présentent des patrons rythmiques très caractéristiques, ce qui est intéressant à analyser et à reproduire. Deuxièmement, cela nous a permis de limiter les détails à considérer lors de la programmation du système et donc de nous concentrer sur la génération. Malheureusement, cela contredit l'aspect générique du système énoncé dans la section 3.1.1. Cela dit, l'architecture du système fait en sorte que les modifications nécessaires pour s'adapter à un autre style sont relativement faciles à effectuer. Ainsi, à la cinquantaine de valses tirées de la base de données Nottingham, nous en avons ajouté d'autres manuellement afin de rétablir l'équilibre entre morceaux en mode majeur et mineur. Cette modification s'est révélée nécessaire pour des raisons expliquées à la section 3.3.

Le corpus remplit deux rôles principaux. Dans un premier temps, il fournit au système un ensemble de morceaux desquels il peut emprunter la suite d'accords et analyser la structure de la mélodie associée, soit pour la répliquer ou pour s'en inspirer. Deux types de patrons sont tirés d'une mélodie du corpus :

— les patrons rythmiques décrivent les durées des notes d'une mesure. Ils sont utilisés pour déterminer les durées des notes de la mélodie générée en assignant un patron rythmique appris dans le corpus à chaque mesure ;

— les patrons structurels décrivent une séquence qui se répète plusieurs fois dans une mélodie. Il peut s'agir d'une séquence de hauteurs, d'intervalles, de durées ou bien une combinaison des trois. Ils ne sont pas limités à une mesure.

Ensuite, l'analyse du corpus au complet permet de récolter des statistiques sur les hauteurs des notes de toutes les mélodies qui sont ensuite utilisées pour mettre au point une heuristique de choix de valeurs dans la seconde phase de résolution.

3.1.4 Algorithme de détection de patrons

L'algorithme utilisé pour identifier les patrons répétitifs dans les mélodies du corpus repose principalement sur l'utilisation des arbres des suffixes, décrits à la section 2.4. Pour pouvoir détecter des patrons sur toutes les dimensions d'une pièce, la mélodie analysée est décomposée en plusieurs chaînes : les hauteurs des notes, les intervalles de hauteurs, les intervalles de hauteurs non qualifiés [1] (pour pouvoir détecter des transpositions même si le nombre de demi-tons n'est pas exactement le même), les durées, les intervalles de durées, les durées avec hauteurs égales consécutives fusionnées et les silences. Les mélodies sont analysées au format

1. aucune différence entre tierce mineure et tierce majeure, entre quinte juste, quinte augmentée et quinte diminuée, etc....

MIDI pour profiter du format numérique mais il est nécessaire d'utiliser le format abc pour calculer les intervalles non qualifiés.

L'algorithme [44] construit un arbre des suffixes pour chaque chaîne en temps linéaire en utilisant l'algorithme d'Ukkonen [45]. Dans chaque arbre, notre objectif est alors de trouver toutes les sous-chaînes d'au moins l caractères qui se répètent au moins x fois. Autrement dit, trouver tous les patrons d'au moins l notes qui présentent au moins x occurrences. Pour cela, on ajoute une information dans l'arbre : on stocke dans chaque nœud la longueur de la sous-chaîne qu'il décrit. L'algorithme se résume alors en un simple parcours d'arbre durant lequel, pour chaque nœud qui présente une longueur supérieure ou égale à l, on note cette longueur puis on compte le nombre de feuilles du sous-arbre dont le nœud en question est la racine. Si il y a au moins x feuilles, cela signifie que la sous-chaîne apparaît au moins x fois. Chaque feuille contient également l'information sur la longueur du suffixe qu'elle décrit. Ainsi, en soustrayant cette longueur à la longueur totale de la chaîne, on obtient l'indice de départ du suffixe. De cette façon, il est facile de trouver les positions de départ de chaque occurrence de chaque patron identifié.

Pour une mélodie de n notes, la construction de l'arbre présente une complexité en $O(n)$ avec l'algorithme d'Ukkonen. Pour l'identification des patrons, un parcours en profondeur permet d'aller chercher les données nécessaires sur les feuilles tout en les comptant, afin de remonter toutes ces informations vers les nœuds parents. Cela permet de savoir pour chaque nœud quelle est la longueur de la chaîne qu'il représente ainsi que combien de fois et à quelles positions elle se répète. Les nœuds pertinents sont alors identifiés. Le parcours se fait donc en temps linéaire dans le nombre de nœuds de l'arbre qui est au maximum $2n + 1$ pour une chaîne de longueur n. Ainsi, la complexité est également de $O(n)$.

Par la suite, il est nécessaire de filtrer les patrons pour éliminer ceux qui sont des suffixes d'autres patrons et ceux qui sont dominés par des plus pertinents. Par exemple, un patron sur les hauteurs de notes implique nécessairement un patron aux mêmes positions sur les intervalles, il n'est donc pas nécessaire de prendre en compte ce dernier.

Techniquement, l'algorithme d'identification des patrons se résume donc en un simple parcours en profondeur avec transmission d'informations des feuilles aux nœuds internes de l'arbre. Le principe de fonctionnement est similaire à ce qui est fait dans [32]. De plus, il s'agit uniquement d'un outil utilisé pour accompagner le système de génération de mélodies décrit par ce travail, il ne fait pas partie intégrante du système. Pour toutes ces raisons, l'algorithme ne constitue pas une contribution importante, c'est pourquoi il est simplement résumé plutôt que détaillé formellement.

Exemple 3.1 (Exemple d'application de l'algorithme)**.** La figure 3.3 présente l'arbre des

suffixes des hauteurs des notes d'une petite mélodie de trois mesures. Autrement dit, l'arbre des suffixes de la chaîne de caractères $ABcdccdcfed$ au format abc. Le symbole \$ représente le caractère de fin de chaîne et on a indiqué sur chaque nœud la longueur de la sous-chaîne correspondante. Si on cherche les patrons d'au moins deux notes qui se répètent au moins deux fois, on trouve les nœuds en bleu et en rouge. En effet, ces deux nœuds correspondent à des sous-chaînes de longueurs 2 et 3 respectivement et possèdent chacun deux feuilles, donc deux occurrences. Cependant, on remarque sur la partition de la mélodie que le patron en bleu n'est qu'un suffixe du patron en rouge et qu'il n'apporte aucune information supplémentaire, on peut donc l'éliminer.

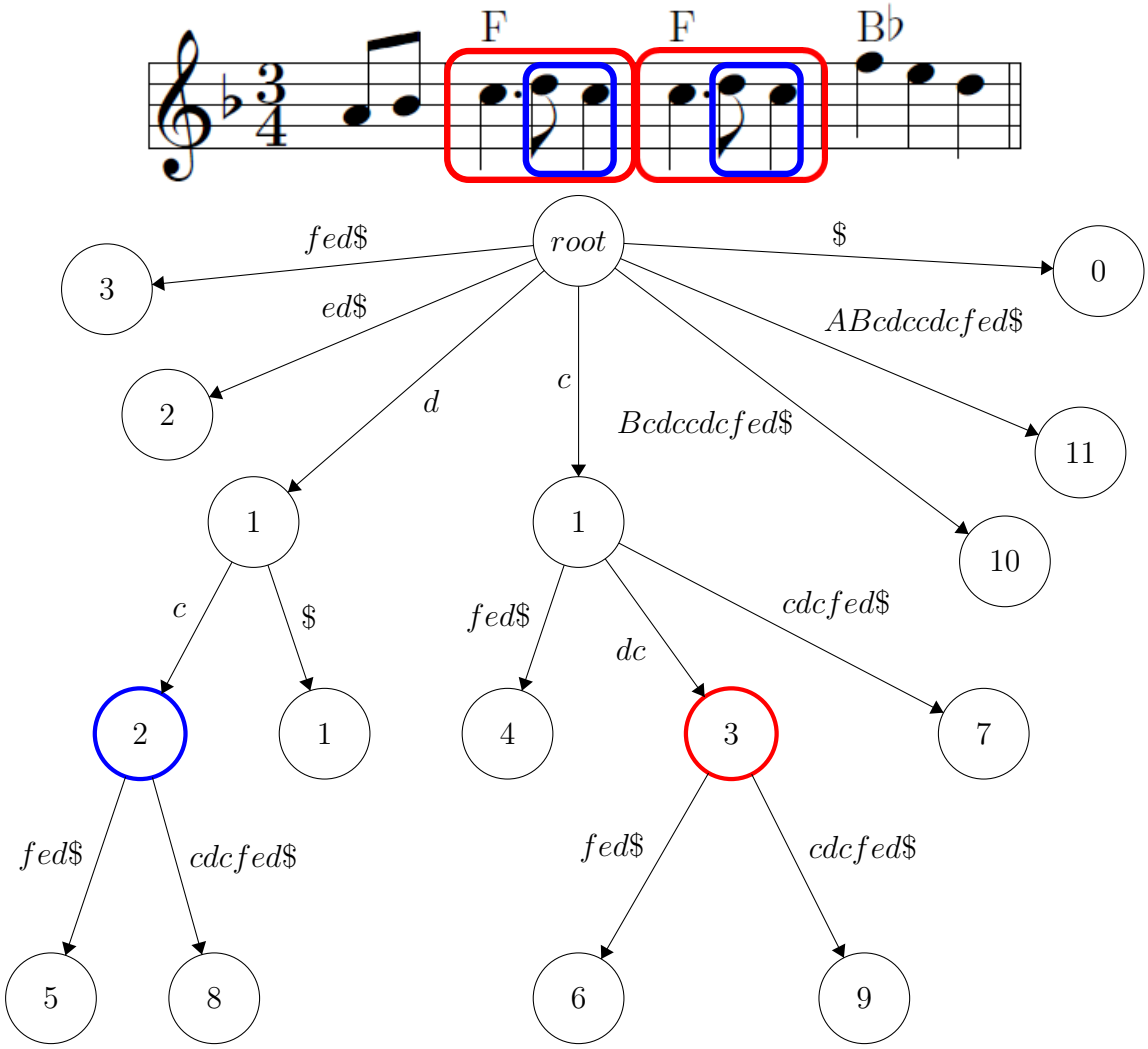


Figure 3.3 Exemple d'application de l'algorithme

La prochaine étape consiste à combiner les patrons des différentes dimensions pour obtenir

des patrons globaux de différents niveaux de pertinence. Dans la version actuelle du système, seulement trois dimensions sont traitées : les hauteurs, les intervalles non qualifiés et les durées. Cela permet de définir trois niveaux de pertinence :

— les patrons fortement pertinents représentent des égalités parfaites sur les trois dimensions entre chaque occurrence d'un patron (hauteurs (et donc intervalles) et durées) ;

— les patrons moyennement pertinents représentent des transpositions. Leurs intervalles et leurs durées sont identiques entre deux occurrences, mais pas leurs hauteurs ;

— les patrons de pertinence faible n'impliquent que les durées des notes, ils n'influent donc que sur le rythme de la mélodie.

Ces trois niveaux vont permettre au système de savoir quelles sont les contraintes à mettre en place en vérifiant la pertinence du patron.

Après toutes ces manipulations, l'algorithme sort une liste des patrons au format texte, triés par niveau de pertinence et par ordre d'apparition.

3.1.5 Fonctionnement du système

Le système prend en entrée une mélodie du corpus qu'on appellera la mélodie d'origine. Les informations importantes de celle-ci sont : la liste des patrons structurels fournie par l'algorithme de la section précédente, sa suite d'accords ainsi que ses patrons rythmiques et leurs fréquences d'apparition. L'utilisateur fournit également au système les contraintes additionnelles qu'il veut poser. Il peut choisir de modifier les patrons structurels, de répertorier les patrons rythmiques du corpus au complet plutôt que de se limiter à la mélodie d'origine et il peut également ajouter des contraintes sur les durées ou les hauteurs des notes. Les détails techniques sont expliqués dans l'annexe A.

Le système va donc se servir des informations sur les patrons rythmiques lors de la première phase durant laquelle il va fixer le nombre et les durées des notes de la mélodie. Puis, dans la seconde phase, il va se servir des statistiques sur les hauteurs des notes du corpus au complet pour guider sa recherche et fixer celles de la mélodie.

La mélodie générée est sortie aux formats MIDI (incluant les accords) et abc ainsi qu'au format PDF si le programme `abcm2ps` est installé.

3.2 Première phase

La première phase de résolution a originellement été introduite pour fixer le nombre de notes de chaque bloc de notes afin de pouvoir créer les tableaux de variables. Elle a donc la

responsabilité de créer la structure en blocs. Dans la version actuelle du système, elle joue un rôle plus important : elle fixe à la fois les longueurs des blocs et les durées des notes qui les composent. Pour faire cela, le système utilise les patrons rythmiques tirés du corpus et en assigne un à chaque mesure de la mélodie à générer.

3.2.1 Apprentissage des patrons rythmiques

Comme évoqué précédemment, la valse présente des caractéristiques rythmiques spécifiques. Nous avons décidé de nous servir de cette propriété pour fixer les durées des notes de notre mélodie. Au lieu d'avoir une variable de branchement par note, on associe à chaque mesure un patron rythmique appris du corpus à notre disposition, ce qui a pour effet de fixer à la fois le nombre de notes par mesure et les durées associées.

L'apprentissage des patrons rythmiques se fait de la manière la plus simple possible. Le format de notation musicale abc inclut les positions des barres de mesure. En parcourant le corpus dans ce format, on peut relever les patrons rencontrés puis noter leurs fréquences d'apparition. L'utilisateur du système peut choisir d'utiliser les fréquences du corpus au complet ou de se limiter à la valse d'origine utilisée, comme dans la figure 3.4.

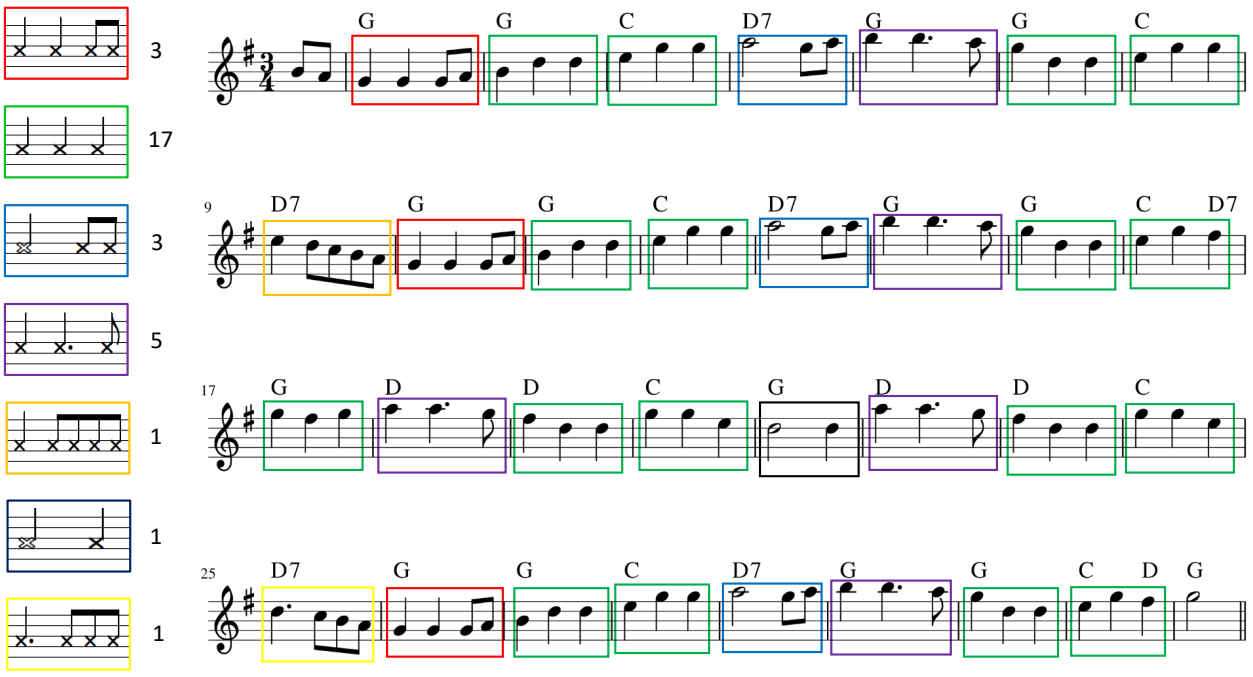


Figure 3.4 Apprentissage des patrons rythmiques

3.2.2 Apprentissage des patrons structurels

L'apprentissage des patrons structurels de la mélodie d'origine se fait grâce à l'algorithme de la section 3.1.4. Ce dernier produit un fichier qui est lu par le système dès la première phase. Les blocs de patrons sont ensuite instanciés, un par occurrence, en les positionnant grâce à leurs temps de départ et à leurs durées. Une fois les blocs créés, les variables couvertes y sont ajoutées ; il devient donc facile de poser les contraintes propres à chaque niveau de pertinence. La figure 3.5 montre un exemple de patron pour chaque niveau de pertinence.

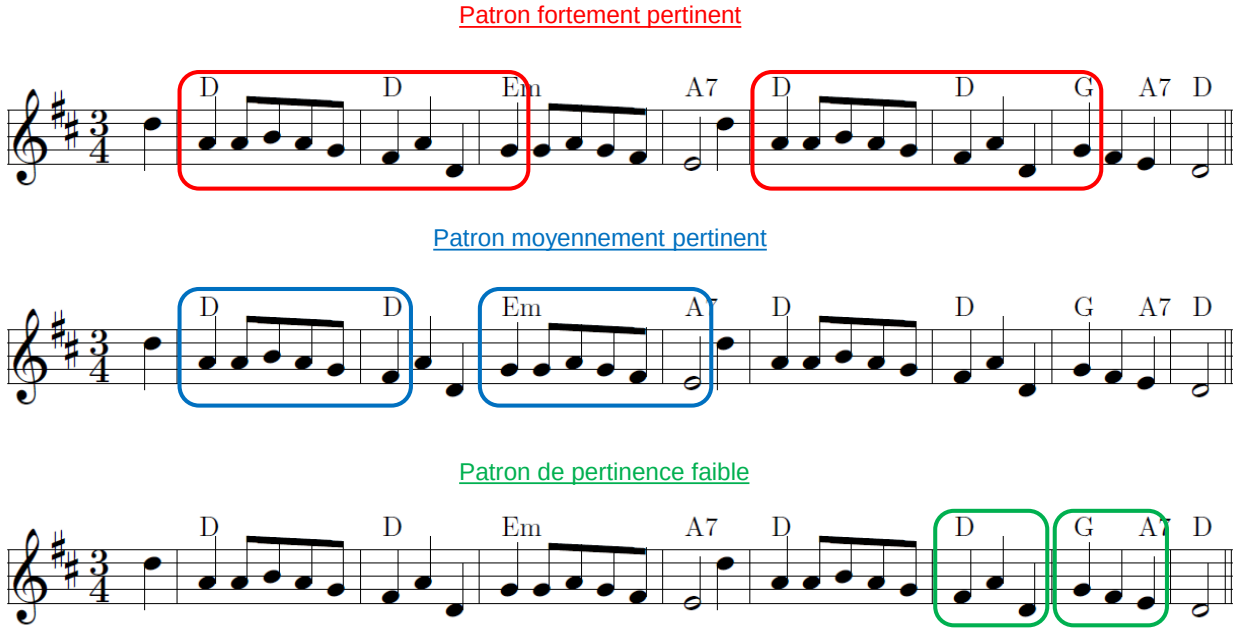


Figure 3.5 Exemples de patrons structurels de différents niveaux de pertinence

3.2.3 Contraintes

La première phase de résolution assigne à chaque mesure un patron rythmique qui fixe à la fois le nombre de notes et leurs durées. Cela est rendu possible par la structure hiérarchique de notre système qui nous permet d'accéder directement à n'importe quelle mesure de la mélodie à générer. Pour assurer la cohérence des patrons structurels qui impliquent les durées des notes, il est nécessaire de mettre en place une contrainte TABLE pour chaque paire de mesures impliquée dans deux occurrences d'un même patron structurel. Ces contraintes spécifient pour chacune de ces paires, quelles sont les paires de patrons rythmiques qui respectent les exigences des patrons structurels. La figure 3.6 montre un exemple de ce type de contrainte

en utilisant les patrons rythmiques de la figure 3.4 dans le cas où la disposition des patrons structurels impose que les deux premiers temps de deux mesures présentent les mêmes durées (patron en vert) et que leur deuxième temps présente la même hauteur de note (patron en rouge) .

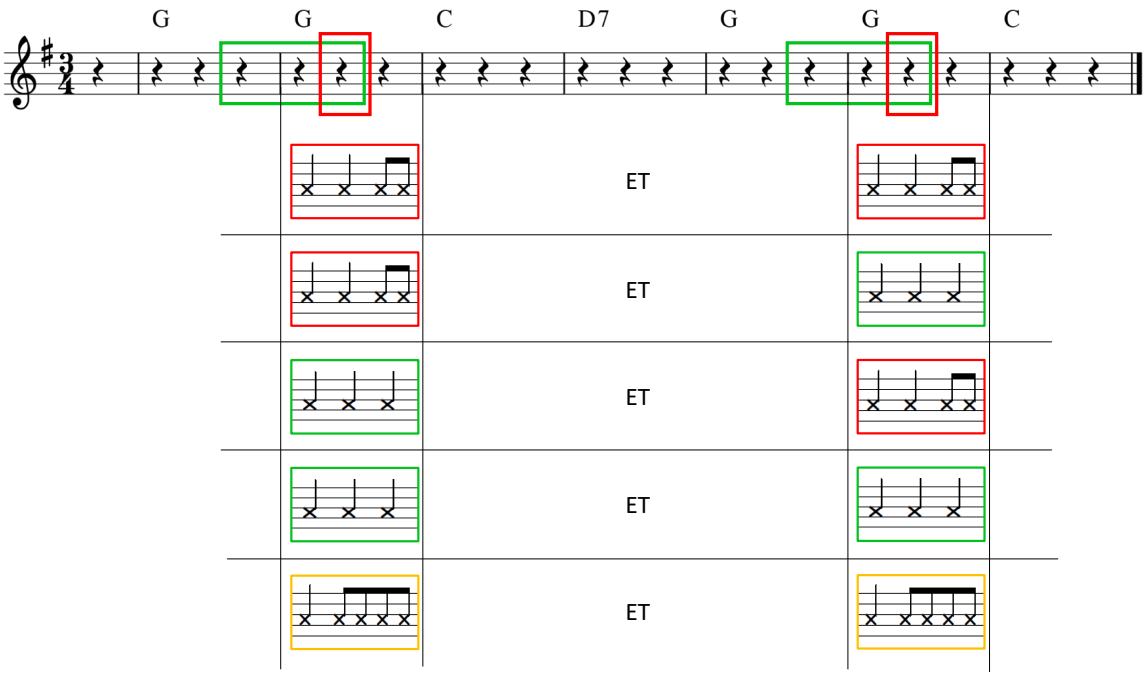


Figure 3.6 Paires de patrons rythmiques possibles selon la disposition des patrons structurels

L'utilisateur peut aussi modifier certains paramètres et choisir d'ajouter des contraintes pour influencer la mélodie générée. Tout d'abord il a le choix de se limiter aux patrons rythmiques de la valse d'origine ou alors de prendre ceux du corpus au complet. Il peut également modifier la pertinence des groupes de patrons structurels. Par exemple, faire en sorte que les patrons de pertinence moyenne agissent comme des patrons forts. Il peut aussi choisir de désactiver une catégorie de patrons. Ces paramètres sont configurables en modifiant les fichiers d'entrée fournis au système (voir l'annexe A pour les détails).

De plus, l'utilisateur possède un contrôle total sur la génération de la structure rythmique. Il peut choisir quel patron rythmique sera assigné à quelle mesure et même imposer une certaine durée de note pour une note en particulier. Il a également la possibilité de poser des contraintes d'égalité entre deux mesures ou groupe de mesures ou entre deux notes ou groupes de notes ainsi que de contrôler le nombre d'occurrences d'un certain patron rythmique ou d'une certaine durée de note. Une contrainte de cardinalité spéciale permet également d'im-

poser une certaine proportion des patrons rythmiques de la valse d'origine lorsque l'utilisateur a choisi d'utiliser ceux du corpus. Pour mettre en place ces contraintes, l'utilisateur doit les fournir au système par le biais d'un fichier texte dans lequel les contraintes sont exprimées grâce à un petit langage détaillé à l'annexe A.4.1. Grâce à la structure hiérarchique des variables du système, l'utilisateur peut accéder directement aux mesures et aux notes sur lesquelles il veut poser ses contraintes. Avec toutes ces possibilités, l'utilisateur peut ainsi mettre en place ses propres patrons et ajuster à son goût le rythme de la pièce à générer.

D'un point de vue technique, pour traduire les exigences de l'utilisateur sous forme de contraintes, le système met en place des contraintes TABLE. Par exemple, si l'utilisateur indique qu'il souhaite que la première note des mesures 1 et 3 aient la même durée, le système détermine la liste de toutes les paires de patrons rythmiques qui respectent cette particularité et impose à la paire de mesures 1 et 3 de prendre leurs patrons rythmiques dans cette liste. De même, si l'utilisateur exige que la troisième note de la mesure 5 soit une croche, le système va dresser la liste des patrons rythmiques dont la troisième note est une croche et forcer la mesure 5 à choisir un patron dans cette liste. Encore une fois, c'est l'organisation hiérarchique du système qui permet d'accéder directement à la mesure voulue et de poser facilement des contraintes de cette nature.

3.2.4 Résolution

La première phase de résolution se charge d'assigner à chaque mesure un patron rythmique. Cela a pour effet de fixer à la fois le nombre de notes de chaque mesure ainsi que leurs durées. Le CSP mis en place est donc le suivant :

— **Variables** : une variable par mesure de la mélodie à générer
— **Domaines** : les patrons rythmiques appris du corpus
— **Contraintes** : les contraintes TABLE qui s'assurent que les durées des notes respectent les patrons structurels mis en place + les contraintes utilisateur

Pour arriver à une solution, le système utilise l'heuristique de choix de variable *minSize* mentionné à la section 1.2.3. Cet ordre va faire en sorte que la résolution suive les patrons structurels puisque ce sont eux qui causent le plus de contraintes (les contraintes TABLE) et donc qui réduisent le plus les domaines des variables de branchement. Ainsi, la résolution commencera généralement par la mesure faisant partie du plus grand nombre de patrons, plus précisément par celle dont le découpage en bloc de notes est le plus restrictif.

Pour les choix de valeurs, le système tire profit des fréquences d'apparition des patrons rythmiques. L'heuristique utilisée choisit un patron aléatoirement selon la distribution décrite

par les informations tirées de la valse d'origine ou du corpus. Sans aucune contrainte utilisateur, plus un patron est fréquent dans la source, plus il sera représenté dans la valse générée. Dans le cas où l'utilisateur a choisi d'utiliser les patrons rythmiques de la valse d'origine, les domaines des variables ne sont pas limités à ceux-ci. Tous les patrons existants dans le corpus font partie du domaine, seulement, ceux qui n'apparaissent pas dans la valse d'origine ont une fréquence nulle. Si pour une raison quelconque, aucun des patrons rythmiques de la mélodie d'origine ne convient à une mesure en particulier, un autre est choisi uniformément au hasard parmi ceux dont la fréquence est nulle mais qui fonctionnent à cet endroit.

3.3 Deuxième phase

La deuxième phase de résolution a la responsabilité de créer les tableaux de variables en utilisant les résultats de la première phase, puis de fixer les hauteurs des notes de la mélodie. Il y a donc une variable de branchement par note et le système fait usage de statistiques issues du corpus pour tenter de générer une mélodie cohérente, musicalement parlant.

3.3.1 Statistique du corpus

Encore une fois, la technique utilisée pour recueillir les statistiques pour les hauteurs de notes est très simple. En parcourant le corpus au format abc, on relève pour chaque note neuf éléments :

- sa hauteur ;
- la fondamentale de l'accord en cours ;
- la qualification de l'accord en cours (majeur, mineur, septième, augmenté ou diminué) ;
- sa position dans la mesure (temps fort ou temps faible) ;
- sa durée
- la hauteur de la note précédente ;
- la fondamentale de l'accord joué avec la note précédente ;
- la qualification de l'accord joué avec la note précédente ;
- la durée de la note précédente.

Les hauteurs relevées ainsi que les fondamentales des accords sont représentées en nombre de demi-tons par rapport à la tonalité du morceau. Par exemple, si la mélodie générée est en ré majeur, la hauteur d'un ré sera représentée par la valeur zéro et la fondamentale d'un accord de fa par la valeur trois.

Ces neuf informations couvrent plusieurs aspects du contexte musical à respecter. La hauteur de la note associée aux informations sur l'accord permet de se faire une idée de quelle note placer sous quel accord. La position dans la mesure précise les notes à favoriser sur le temps fort. L'information sur la durée de la note peut aider le système à comprendre le concept de note de passage. Finalement, les informations sur la note précédente matérialisent la notion d'intervalle, très importante en musique.

Pour stocker ces statistiques, on construit simplement un tableau à neuf dimensions, une pour chaque caractéristique, puis on incrémente la case correspondant à chaque note que l'on rencontre en parcourant le corpus.

3.3.2 Contraintes

Pour générer les hauteurs de notes en spécifiant le moins de règles musicales possible, le système utilise une contrainte COST-REGULAR dont l'automate est construit en fonction des statistiques recueillies dans le corpus. Les états de l'automate représentent un quadruplet {*hauteur, fondamentale, qualification, position*} qu'il est possible de retrouver dans la solution finale. Plusieurs étapes sont donc nécessaires pour construire l'automate :

1. recenser tous les triplets {*fondamentale, qualification, position*} qui figurent dans la structure rythmique avec accords obtenus en première phase
2. pour chacun de ces triplets, parcourir le corpus et relever le nombre d'occurrences de chaque hauteur de note jouée dans cette configuration. Par exemple, si le triplet est {*sol, majeur, temps faible*}, il faut compter le nombre d'occurrences de chaque hauteur jouée sous un sol majeur sur un temps faible d'une mesure. Chacune des hauteurs relevées forme un quadruplet {*hauteur, fondamentale, qualification, position*} avec le triplet en cours de traitement
3. éliminer les quadruplets dont le nombre d'occurrences est inférieur à un certain seuil (préférence de l'utilisateur). Les quadruplets restants sont les états de l'automate

Une fois que les états sont déterminés, il faut mettre en place les transitions. Pour cela, les étapes suivantes sont nécessaires :

4. pour chaque quadruplet q représentant un état de l'automate, localiser toutes ses occurrences dans le corpus
5. identifier pour chacune d'entre elles le quadruplet p formé par la note précédente et l'accord joué avec elle

6. si p est dans la liste des états de l'automate, ajouter une transition de p à q avec un coût égal à l'opposé du nombre d'occurrences de l'événement "p suivi de q" dans le corpus

Le coût associé à chaque transition est négatif, ce qui permet au système de contrôler la qualité de la solution en imposant une borne supérieure ou en minimisant son coût total. Un état initial est ajouté avec une transition vers tous les autres états pour un coût nul.

Exemple 3.2 (Exemple d'automate simplifié)**.** La figure 3.7 présente un petit corpus de 4 mélodies (à gauche) ainsi que la structure rythmique obtenue en l'utilisant pour la première phase de résolution (à droite).

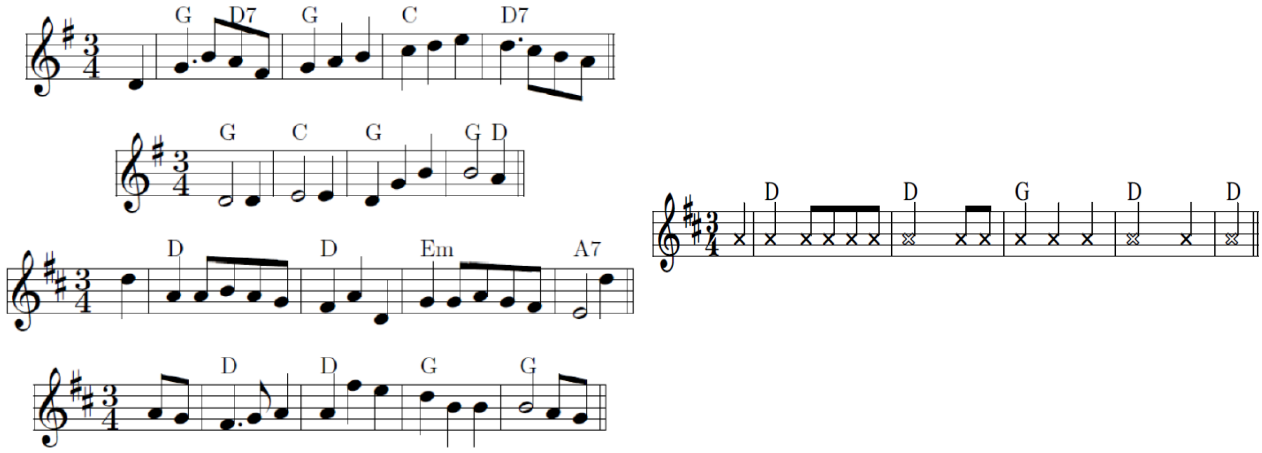


Figure 3.7 Exemple de corpus et de structure rythmique obtenue

En utilisant ces informations, le système construit un automate tel que décrit précédemment qui dans ce cas contient 16 états au total. Pour alléger sa représentation graphique, l'automate de la figure 3.8 est simplifié : les positions dans la mesure sont ignorées et l'état initial n'est pas représenté. Le seuil utilisé est 0.

Malgré notre objectif qui était d'imposer le moins de règles musicales explicites possible au système, nous avons dû ajouter deux contraintes à la seconde phase pour améliorer la qualité des résultats produits. Premièrement, il a fallu forcer la première note sous chaque accord à prendre une hauteur faisant partie de ce dernier pour éviter des dissonances trop prononcées. Grâce à la présence des blocs d'accords dans le système, cette contrainte est exprimée très aisément. Ensuite, nous avons aussi imposé à la toute dernière note de la mélodie d'être égale à la tonique du morceau.

Pour éviter les intervalles trop brusques, nous avons posé une contrainte souple sur les intervalles suivant une note plus courte qu'une noire. Lors de la résolution, le solveur minimise le

coût de la solution pour éviter les intervalles de plus d'une tierce entre ces notes.

Tout comme pour la première phase, l'utilisateur peut aussi ajouter ses propres contraintes d'égalité ou de cardinalité sur les hauteurs des notes de la mélodie.

3.3.3 Résolution

La première phase de résolution a permis de fixer le nombre de notes de la mélodie ainsi que leurs durées. La seconde phase se charge d'assigner une hauteur à chacune de ces notes. Le CSP mis en place est donc le suivant :

— **Variables** : une variable par note de la mélodie à générer

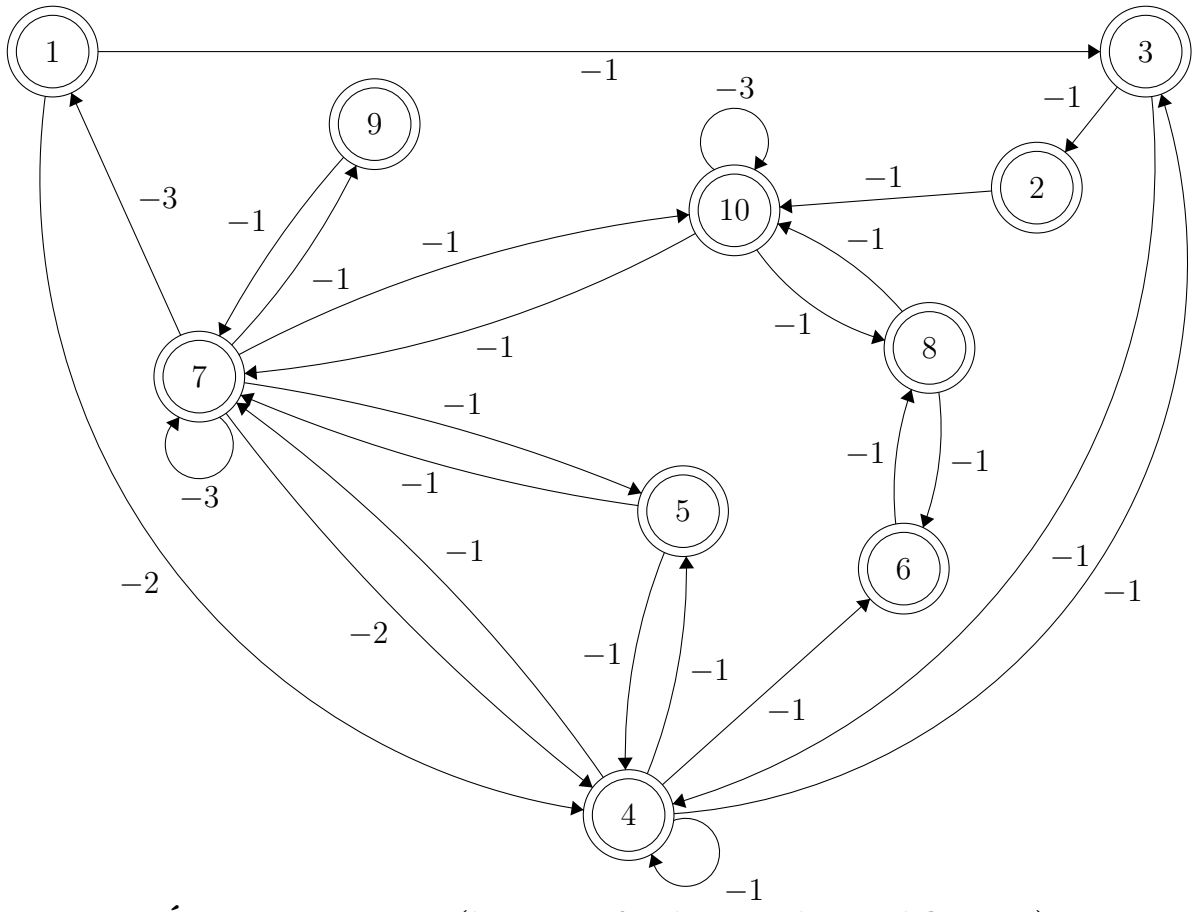
— **Domaines** : les hauteurs possibles pour une note donnée

— **Contraintes** : la contrainte COST-REGULAR + les contraintes du domaine musical + les contraintes utilisateur

L'heuristique de choix de variable utilisée dans la seconde phase est simplement l'ordre lexicographique des variables, autrement dit, les hauteurs des notes sont fixées les unes après les autres, en commençant par la première. Cet ordre permet de profiter des informations à propos des intervalles récoltées dans le corpus. En effet, le système pourra faire un branchement plus précis s'il connaît la hauteur de la note précédant celle qu'il est en train de traiter.

Les statistiques issues du corpus font une nouvelle fois partie intégrante de l'heuristique de choix de valeur pour la seconde phase. À ce point-ci de l'exécution du système, la suite d'accords et les durées de notes sont fixées. Puisqu'on progresse en ordre lexicographique, la seule inconnue parmi les sept dimensions des statistiques à un point donné de la résolution est la hauteur de la note. Il est donc possible de construire un tableau des fréquences qui indique le nombre d'apparitions de chaque hauteur du domaine selon le contexte actuel. Ce tableau fourni la distribution à respecter pour le choix de valeur aléatoire.

Dans le cas où le contexte musical est inconnu et où toutes les fréquences sont nulles, le système crée un nouveau tableau de fréquences en faisant la somme sur les trois dernières dimensions (durée, hauteur précédente et durée précédente) et réessaie de sélectionner une valeur selon cette nouvelle distribution. Si le même problème survient, une valeur est choisie uniformément au hasard dans le domaine de la variable.



État	Description (hauteur, fondamentale, qualification)
1	Ré, Ré, Majeur
2	Ré, Sol, Majeur
3	Mi, Ré, Majeur
4	Fa#, Ré, Majeur
5	Sol, Ré, Majeur
6	Sol, Sol, Majeur
7	La, Ré, Majeur
8	La, Sol, Majeur
9	Si, Ré, Majeur
10	Si, Sol, Majeur

Figure 3.8 Exemple d'automate simplifié. Le coût de la transition d'un état A vers un état B est égal à l'opposé du nombre d'occurrences de l'événement "A suivi de B" dans le corpus

CHAPITRE 4 RÉSULTATS

4.1 Détails techniques

Ce projet a été réalisé en C++ et 8850 lignes de code ont été écrites pour le mener à bien. Le solveur utilisé est ILOG Solver d'IBM. Les résultats décrits dans cette section ont été produits sous CentOS Linux sur une machine composée d'un AMD Opteron 275 cadencé à 2.2 GHz et de 12 Go de mémoire vive. Le projet est hébergé sur le serveur du laboratoire QUOSSEÇA et le code est géré avec le logiciel SVN.

4.2 Description des résultats générés

Les exemples suivants décrivent trois exécutions du système. Les détails sur le format des fichiers d'entrée sont donnés à l'annexe A.

Exemple 4.1. Ce premier exemple utilise les fréquences des patrons rythmiques du corpus plutôt que ceux de la valse d'origine. De plus, les patrons structurels moyennement pertinents sont convertis en patrons de faible pertinence. Seules deux contraintes utilisateur sont ajoutées : l'indice du patron rythmique de la seconde mesure doit être différent de 0 et de 11. Autrement dit, la seconde mesure ne peut pas contenir six croches (patron 0) ou avoir le même patron rythmique que l'origine (patron 11).

L'automate généré par cet exemple comporte 123 états et le problème contient 96 variables de branchement au total. L'annexe B.2 montre les fichiers d'entrée utilisés et la mélodie générée.

Pour dix exécutions avec ces mêmes paramètres, neuf d'entre elles arrivent à la solution optimale (après une recherche exhaustive) en un temps de résolution moyen de 1,18 seconde. Le temps d'exécution moyen quant à lui est de 14,25 secondes. Cette différence est principalement due à la lecture des statistiques du corpus pendant laquelle le système doit remplir un tableau à sept dimensions, ce qui prend beaucoup de temps. La dixième exécution a atteint la limite des quinze secondes sans trouver de solution, puis a recommencé de zéro pour finalement trouver une première solution en une dizaine de secondes et atteindre une nouvelle fois la limite. Le temps de résolution mentionné est à peu de choses près entièrement consacré à la seconde phase, la première étant à chaque fois résolue quasi instantanément.

Exemple 4.2. Comme dans le cas précédent, ce sont les fréquences des patrons rythmiques du corpus qui sont utilisées pour cet exemple. Les patrons structurels de pertinence moyenne sont à nouveau interprétés comme des patrons de pertinence faible, mais cette fois-ci aucune

contrainte utilisateur n'est ajoutée. Par contre, pour montrer ce qui peut arriver lorsque les deux contraintes mentionnées à la section 3.3.2 sont absentes, elles ont été désactivées. Ainsi les premières hauteurs sous chaque accord ne sont plus restreintes aux hauteurs de ce dernier et la dernière note de la mélodie n'est pas forcée d'être la tonique. L'annexe B.3.5 montre la partition générée et on peut voir que la seconde note de la mélodie est un peu gênante. En effet, il s'agit d'un ré joué en même temps qu'un accord de fa majeur ce qui forme une sixte majeure qu'on qualifie de consonance imparfaite.

L'automate de cet exemple précis comporte 69 états et le système contient 117 variables de branchement au total. L'annexe B.3 montre les fichiers d'entrée utilisés.

La recherche de solutions pour cet exemple a semblé plus difficile que pour le précédent. Sur dix exécutions, seules quatre sont arrivées à optimalité avec un temps de résolution moyen de 4,16 secondes et un temps d'exécution moyen de 18,16 secondes. Les six autres ont trouvé une première solution avec un temps de résolution moyen de 8,82 secondes puis ont atteint la limite de temps imposée de quinze secondes. Comme pour l'exemple 4.1, la première phase est à chaque fois résolue moins de 0,03 seconde.

Exemple 4.3. Dans ce dernier exemple, ce sont les fréquences des patrons rythmiques de la valse qui sont utilisées et tous les patrons structurels sont ajoutés avec leur niveau de pertinence original. Aucune contrainte utilisateur n'est ajoutée mais les statistiques sur les hauteurs de notes sont limitées à la mélodie d'origine plutôt qu'à tout le corpus. De plus, un objectif de minimisation du coût total de l'automate COST-REGULAR de la seconde phase a été ajouté. Cette configuration du système permet de générer une mélodie qui reproduit exactement la valse dont elle s'inspire. L'annexe B.4 détaille les fichiers d'entrée utilisés et les résultats obtenus.

4.3 Évaluation des mélodies générées

L'évaluation des créations du système s'est faite selon trois axes de réflexion. Premièrement, est-il nécessaire de définir de nouvelles contraintes dédiées et de spécifier manuellement des règles du domaine musical pour générer des mélodies crédibles en PPC ? Ensuite, l'analyse se porte vers l'extraction des patrons structurels et cherche à savoir si cette méthode permet de résoudre le problème de manque de structure globale d'une mélodie synthétique identifié lors de la revue de littérature. Finalement, le troisième aspect concerne la capacité de création du système.

Pour aborder ces trois questions, en nous inspirant des travaux mentionnés à la section 2.5, nous avons mis au point un sondage en ligne. Trois aspects sont mis en avant : la crédibilité,

la variété et la qualité. Nous avons sélectionné dix valses du corpus pour générer nos résultats.

Pour évaluer la capacité du système à générer des mélodies crédibles, nous avons généré deux extraits avec chacune des dix valses en utilisant leurs fréquences de patrons rythmiques et en répliquant tous les patrons structurels. Nous avons ensuite demandé aux personnes interrogées d'essayer de distinguer la valse d'origine des deux générées pour chaque exemple. Les résultats sont présentés dans le tableau 4.1.

Pour le volet variété, nous avons répété l'expérience mais cette fois-ci en variant la provenance des patrons rythmiques et la pertinence des patrons structurels. De plus, nous avons ajouté des contraintes utilisateur pour générer des mélodies les plus personnalisées possible. Les questions du sondage demandaient aux participants de quantifier sur une échelle de un à cinq à quel point ils trouvaient que nos créations étaient différentes entre elles et différentes des pièces dont elles s'inspirent. Le tableau 4.2 présente les résultats obtenus.

Finalement, pour évaluer la qualité nous avons simplement demandé aux personnes interrogées de noter les extraits que nous avons générés en termes de "plaisir d'écoute" sur une échelle de un à cinq. Les résultats obtenus sont compilés dans le tableau 4.3.

4.4 Analyse

Au total, dix-neuf personnes ont participé au sondage en répondant à un nombre variable de questions. Parmi ces dix-neuf sujets, cinq ont indiqué qu'ils étaient musiciens. Les valses du corpus et donc les valses générées contiennent environ une trentaine de mesures.

Le tableau 4.1 nous informe qu'en moyenne, plus de la moitié des réponses fournies identifient correctement la valse d'origine, ce qui n'est pas en faveur de notre système. Seules deux valses ont réussi à tromper plus de la moitié des personnes qui ont répondu aux questions les concernant. Lors d'une expérience similaire au cours d'une conférence, 58% des personnes interrogées n'avaient pas été capables de distinguer la valse d'origine qui était dans ce cas la numéro 26. Curieusement, c'est cette dernière qui, avec la valse 24, présente les moins bons résultats cette fois-ci. Au moment de la conférence, le système ne fonctionnait pas tout à fait de la même manière que maintenant, ce qui peut expliquer cette différence dans les résultats. En moyenne, plus de quarante pour cent des réponses sont en faveur de notre système ce qui reste tout de même satisfaisant.

À cause de certaines difficultés rencontrées lors de la génération des valses en mode mineur, les extraits dans ce mode qui sont utilisés pour le sondage nous semblaient de moins bonne qualité que ceux générés en mode majeur. Par conséquent, nous nous attendions à ce que, pour les valses en mode mineur, les participants identifient plus facilement la valse d'origine.

Tableau 4.1 Résultats du sondage pour l'évaluation de la crédibilité. Les pourcentages indiquent la proportion de réponses qui ont identifié une valse créée par le système comme valse d'origine.

Valse d'origine	Tonalité	Global	Non-musiciens	Musiciens
8	F maj	64,29%	54,55%	100,00%
9	D maj	27,27%	11,11%	100,00%
12	G maj	64,29%	70,00%	50,00%
15	D min	46,15%	45,45%	33,33%
22	D maj	35,71%	27,27%	66,67%
24	D maj	23,08%	20,00%	33,33%
26	G maj	25,00%	20,00%	50,00%
46	A min	42,86%	50,00%	25,00%
68	A min	46,15%	55,56%	25,00%
93	E min	38,46%	40,00%	33,33%
Moyenne	min	43,40%	47,50%	28,57%
Moyenne	maj	41,03%	34,43%	64,71%
Moyenne	global	41,98%	39,60%	48,39%

Pourtant, globalement, les résultats sont pratiquement identiques entre les deux modes. Le comportement attendu se retrouve quand même chez les sujets musiciens. Cependant, ces derniers ont été bien moins performants que les autres participants. Une des explications possibles est le faible nombre d'individus dans ce groupe qui fait en sorte que les résultats ne sont pas aussi fiables qu'ils pourraient l'être.

D'après les résultats présentés dans le tableau 4.2, notre système est capable de générer des mélodies significativement différentes de celles dont il s'inspire. Avec un score moyen de 3,48 sur une possibilité de 5, nous pouvons affirmer que l'objectif est atteint sur ce point. Cependant, le degré de différence moyen entre deux valses générées à partir d'une même mélodie est légèrement inférieur. Ce résultat peut s'expliquer par la petite taille du corpus, qui fournit donc des statistiques limitées pour la génération. Ainsi, même si deux générations présentent des paramètres et des contraintes différents, les résultats peuvent être plus similaires que prévu. Une autre option pour améliorer cet aspect serait de fournir plus d'options à l'utilisateur. Ce point est évoqué à la section 5.2.

L'appréciation moyenne des créations de notre système est de 3,29 sur 5, comme le montre le tableau 4.3. Un résultat tout à fait raisonnable si on le compare au degré de qualité attribué aux valses d'origine par les individus interrogés qui est de 3,50. Encore une fois, nous nous attendions à une bien moins bonne qualité pour les valses en mode mineur, les résultats obtenus dans ce cas dépassent donc nos attentes. Les extraits générés à partir des valses 9

Tableau 4.2 Résultats du sondage pour l'évaluation de la variété. Les scores indiquent le niveau de différence sur une échelle de 1 (identique) à 5 (complètement différent).

Valse d'origine	Tonalité	Différence générées/origine	Différence générées/générées
8	F maj	3,50	2,83
9	D maj	3,33	3,33
12	G maj	3,17	2,83
15	D min	2,92	2,67
22	D maj	3,79	4,00
24	D maj	3,83	3,50
26	G maj	3,75	3,50
46	A min	3,92	3,17
68	A min	3,38	3,00
93	E min	3,10	3,40
Moyenne	min	3,33	3,04
Moyenne	maj	3,54	3,31
Moyenne	global	3,46	3,20

et 22 ont été mieux notés que ces dernières, ce qui montre que notre système est capable de composer des mélodies que les gens apprécient.

En mettant en relation les tableaux 4.2 et 4.3, une certaine corrélation apparaît entre les résultats générés à partir de la valse 46. En effet, ces mélodies sont à la fois celles qui sont les plus différentes de leur origine et celles qui sont le moins appréciées. La valse 46 du corpus est dans la tonalité de La mineur. Les difficultés que nous avons éprouvées avec le mode mineur se font donc ressentir ici, car plus on essaye de s'éloigner de l'origine, plus on doit inclure des notes peu élégantes.

On remarque également que la valse d'origine 15, elle aussi en mineur, est celle qui est la plus similaire aux extraits dont elle est la source d'inspiration et que ceux-ci sont aussi les moins différents entre eux. Encore une fois, ce sont les problèmes rencontrés avec les morceaux dans cette tonalité qui nous ont forcés à réduire le nombre de contraintes utilisateur imposées pour être capables de générer une mélodie musicalement raisonnable.

Tableau 4.3 Résultats du sondage pour l'évaluation de la qualité. Les scores indiquent le niveau de plaisir éprouvé par les auditeurs sur une échelle de 1 (plus jamais) à 5 (je l'ajouterais à ma playlist).

Valse d'origine	Tonalité	Qualité de l'origine	Qualité des valses générées
8	F maj	3,50	2,75
9	D maj	3,83	3,92
12	G maj	3,67	3,08
15	D min	3,33	3,17
22	D maj	2,86	3,71
24	D maj	3,50	3,42
26	G maj	3,50	3,10
46	A min	3,50	2,67
68	A min	3,50	3,50
93	E min	4,00	3,50
Moyenne	min	3,57	3,20
Moyenne	maj	3,53	3,30
Moyenne	global	3,54	3,26

4.5 Discussion

4.5.1 A-t-on besoin de nouvelles contraintes dédiées et de règles explicites pour générer des mélodies crédibles ?

Les résultats décrits dans le tableau 4.1 montrent que les mélodies générées ne sont pas aussi crédibles que nous l'espérions mais restent tout de même relativement convaincantes : plus de quarante pour cent des réponses récoltées identifient une valse du système comme composée par un être humain. Le test de Turing effectué présente cependant un léger défaut qui pourrait expliquer cette performance moyenne. En effet, puisque les deux mélodies générées utilisent des informations tirées d'une seule et même pièce, les personnes interrogées auraient pu les identifier en remarquant qu'elles étaient plus différentes entre elles que par rapport à l'origine (le tableau 4.2 confirme cette constatation).

Ces résultats raisonnables combinés aux commentaires des personnes interrogées qui ont souvent relevé la difficulté de distinguer les mélodies du système des mélodies d'origine montrent que nous avons atteint notre objectif sur ce point. Les résultats des tests de qualité recensés dans le tableau 4.3 appuient cette interprétation.

La raison principale de ce succès est la structure des variables du système. Notre organisation hiérarchique nous permet d'exprimer les contraintes nécessaires sans avoir besoin d'en

développer de nouvelles comme ont dû le faire les chercheurs du Sony CSL Paris avec les contraintes ALLEN et METER. Ainsi, nous avons profité des similarités entre la PPC et la musique pour mettre au point un système de contraintes musical dont le fonctionnement met à profit ce parallèle et qui propose donc un fonctionnement instinctif.

Un des inconvénients liés à l'utilisation de données concrètes est la difficulté de représenter la notion de style. Les patrons rythmiques et les statistiques que nous recueillons dans le corpus nous servent de guide pour la génération mais ne véhiculent peut-être pas la même qualité ni quantité d'informations stylistiques que les modèles plus complexes comme les chaînes de Markov par exemple. Cette limitation pourrait être une des explications pour les résultats de la section crédibilités du sondage qui sont moins bons qu'anticipés.

4.5.2 L'extraction des patrons structurels permet-elle de résoudre le problème du manque de structure globale ?

L'utilisation des arbres des suffixes pour la détection des patrons répétitifs s'inscrit également dans notre volonté d'utiliser des méthodes simples et explicites plutôt que des algorithmes d'apprentissage ou de forage de données. Les informations extraites des mélodies existantes nous permettent d'imposer une structure globale aux mélodies générées par notre système, en tirant profit encore une fois de notre organisation hiérarchique.

Les mélodies générées présentent une structure cohérente inspirée de celles du corpus. Les résultats présentés en annexe B le montrent bien. Les résultats relatifs à l'appréciation des créations de notre système nous confortent dans la validation de nos objectifs en termes de structure mélodique. En effet, on retrouve constamment des structures répétitives dans la musique populaire moderne, cet aspect est donc directement lié à la satisfaction de l'auditeur par rapport à la mélodie.

L'algorithme en lui-même, notamment la combinaison des patrons de différentes dimensions, pourrait néanmoins être optimisé. De plus, la traduction des patrons trouvés sous forme de contraintes aurait avantage à être légèrement modifiée pour rendre le tout plus flexible. Il serait par exemple possible d'identifier le début et la fin des occurrences par leur position dans la mesure (temps fort ou temps faible) plutôt que leur temps de début et de fin absolus. En utilisant ces repères temporels stricts, le système force les blocs de patrons de la mélodie générée à commencer (ou finir) à un moment précis. Dans le cas où celui-ci se trouve au milieu du premier temps d'une mesure par exemple, cela filtre automatiquement tous les patrons rythmiques dont le premier temps n'est pas divisé en au moins deux croches. Cette modification permettrait donc aux rythmes des mélodies générées d'être plus variés puisqu'il y aurait alors, en règle générale, plus de patrons rythmiques possibles pour les mesures aux

extrémités des blocs de patrons.

4.5.3 Le système mis en œuvre est-il un bon outil de créativité musicale ?

Les résultats présentés dans les tableaux 4.1, 4.2 et 4.3 montrent que dans sa version actuelle, le système est capable de générer des mélodies relativement crédibles, variées et appréciées. Additionnellement, le support des contraintes utilisateur permet de créer des phrases musicales personnalisées. Avec quelques fonctionnalités supplémentaires, le système pourrait être utile pour la génération de courtes mélodies libres de droits par exemple. Grâce à la flexibilité de la structure hiérarchique et de la PPC, il peut également être utilisé pour compléter une mélodie existante en cas de manque d'inspiration.

Le système présenté dans ce travail est basé sur une logique simple intuitive qui permet de facilement ajouter des fonctionnalités et de personnaliser les résultats générés. Chaque utilisateur peut l'ajuster selon ses besoins et ainsi créer des mélodies variées et agréables. Le système constitue donc la base d'un bon outil de créativité musicale.

CHAPITRE 5 CONCLUSION

Ce mémoire a présenté les travaux de conception et d'implémentation d'un système de génération automatique de mélodie basé sur la programmation par contraintes. Après une revue de la littérature du domaine, nous avons décidé de nous concentrer sur deux aspects en particulier : l'organisation des variables et la structure de la mélodie générée. Les chapitres précédents ont donc décrit l'architecture du système et les moyens mis en œuvre pour atteindre nos objectifs.

5.1 Synthèse

Le système est bâti sur un modèle hiérarchique. Nous avons émis l'hypothèse qu'une organisation des variables de cette nature favorise l'expression des contraintes et contribue donc à la flexibilité du système. De plus, la musique est elle-même basée sur des relations hiérarchiques entre la mélodie, les mesures, les accords.... Ce choix engendre par contre une difficulté au niveau de la résolution puisque le nombre de variables du problème devient lui-même une variable. Le système est donc forcé de fonctionner en deux phases de résolution afin de pouvoir fixer dans un premier temps le nombre et la durée des notes puis les hauteurs dans un second temps.

Pour générer une mélodie, le système dépend d'un corpus duquel il utilise les suites d'accords et tire profit des statistiques sur les patrons rythmiques et les hauteurs de notes pour mettre au point des heuristiques de choix de valeurs. Ainsi lors de la première phase, il assigne à chaque mesure un patron rythmique qui fixe alors le nombre de notes de ladite mesure ainsi que leurs durées. Le choix de valeurs se fait de manière aléatoire selon la distribution décrite par les nombres d'occurrences des patrons rythmiques de la valse d'origine ou bien du corpus au complet.

La seconde phase de résolution est basée sur une contrainte COST-REGULAR qui contrôle la séquence de hauteurs de notes. L'automate est construit selon les statistiques recueillies du corpus et permet de s'assurer que la mélodie reste cohérente musicalement. Deux contraintes dures viennent compléter la modélisation de la deuxième phase afin d'éviter des dissonances gênantes qui peuvent survenir à cause de la nature aléatoire de l'heuristique de choix de valeurs.

Pour répliquer la structure d'une mélodie du corpus, nous avons utilisé un algorithme simple utilisant des arbres des suffixes pour analyser chacune de ses dimensions. L'algorithme trouve

les séquences qui se répètent et combine les résultats obtenus dans chaque dimension pour former des patrons de différents niveaux de pertinence. Ces niveaux déterminent quelles contraintes doivent être posées entre deux occurrences d'un patron structurel.

Lorsque le système est utilisé pour imiter une mélodie donnée, cette dernière fournit ses patrons rythmiques, sa suite d'accords ainsi que tous ses patrons structuraux qui sont alors répliqués dans le résultat. Pour générer une mélodie originale, l'utilisateur a la possibilité de modifier la provenance des patrons rythmiques ainsi que le niveau de pertinence des patrons structuraux. Il peut également poser des contraintes additionnelles sur les notes et les dimensions de son choix.

5.2 Limitations

La structure hiérarchique des variables résout les problèmes identifiés dans la revue de littérature mais engendre également quelques limitations. La division de la résolution du problème en deux phases empêche de profiter des mécanismes de *backtracking* intégrés au solveur entre les variables de la première phase et celles de la seconde. Cela nous a forcé à ajouter des contraintes à la première phase pour prendre en compte des informations à propos de la seconde. Cependant nous n'avons pas été en mesure d'anticiper toutes les possibilités et il arrive parfois, lorsqu'on essaye de générer une mélodie personnalisée, que les durées des notes ne correspondent pas aux statistiques et donc que l'automate de la phase deux n'admette pas de solutions.

Ces problèmes sont également liés au corpus et nous pensons que ce dernier est la principale limitation du système. En effet, en nous limitant aux valses nous avons réduit la taille du corpus à une centaine de chansons, ce qui ne produit pas des statistiques assez denses pour avoir un système robuste. De plus, le déséquilibre entre pièces en mode majeur et mineur dans le corpus a un impact important sur le système puisque cela signifie que le seuil utilisé pour construire l'automate est différent selon le mode de la mélodie d'origine.

5.3 Améliorations futures

Si nous devions continuer les travaux sur ce système, la première étape serait de ne plus le limiter aux valses et de le rendre plus robuste. En modifiant légèrement le système pour traiter toutes les signatures temporelles, il serait alors facile d'augmenter considérablement la taille du corpus et donc de récolter des statistiques plus denses. Cela aiderait beaucoup pour être capable de générer toutes sortes de mélodies et expérimenter avec des corpus de différents styles.

En ce qui concerne l'algorithme de détection des patrons, comme mentionné à la section 4.5, le processus de combinaison des dimensions pourrait être amélioré et l'application des patrons structurels plus flexible en général. Il serait également intéressant d'incorporer les informations des dimensions que nous avons ignorées jusqu'à maintenant (intervalles de durées, durées avec hauteurs égales consécutives fusionnées et silences). De plus, pour faciliter l'utilisation du système, il faudrait automatiser l'exécution de l'algorithme et la création des fichiers d'entrée selon la valse d'origine choisie. Pour l'instant, l'algorithme de détection des patrons structurels et la récolte des statistiques du corpus doivent être exécutés séparément par l'utilisateur qui doit aussi créer manuellement la suite d'accords et y ajouter les fréquences des patrons rythmiques.

Les heuristiques de choix de valeur pour les deux phases pourraient possiblement être améliorés en incorporant les informations sur le degré des variables et ainsi fixer en premier celles qui sont impliquées dans le plus de contraintes et donc dans le plus d'occurrences de patrons structurels.

L'objectif à long terme est de fournir une interface graphique à l'utilisateur pour simplifier l'utilisation du système mais aussi augmenter la flexibilité et les possibilités de création.

RÉFÉRENCES

[1] D. Herremans, C.-H. Chuan, and E. Chew, "A functional taxonomy of music generation systems," *ACM Computing Surveys (CSUR)*, vol. 50, no. 5, p. 69, 2017.

[2] J. D. Fernández and F. Vico, "Ai methods in algorithmic composition : A comprehensive survey," *Journal of Artificial Intelligence Research*, vol. 48, pp. 513–582, 2013.

[3] S. Demassey, G. Pesant, and L.-M. Rousseau, "A cost-regular based hybrid column generation approach," *Constraints*, vol. 11, no. 4, pp. 315–333, 2006.

[4] G. Pesant, "A regular language membership constraint for finite sequences of variables," in *International conference on principles and practice of constraint programming*, pp. 482–495, Springer, 2004.

[5] IBM, *IBM ILOG Solver Reference Manual*, 2009.

[6] T. Anders and E. R. Miranda, "Constraint programming systems for modeling music theories and composition," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 30, 2011.

[7] L. A. Hiller and L. M. Isaacson, *Experimental Music : Composition With an Electronic Computer*. McGraw-Hill, 1959.

[8] K. Ebcioglu, "Computer counterpoint," in *Proceedings of the 1980 International Computer Music Conference, ICMC 1980, New York City, USA, 1980*, 1980.

[9] B. Schottstaedt, "Automatic species counterpoint," Tech. Rep. STAN-M-19, CCRMA, Department of Music, Stanford University, May 1984.

[10] K. Ebcioğlu, "An expert system for harmonizing chorales in the style of js bach," *The Journal of Logic Programming*, vol. 8, no. 1-2, pp. 145–185, 1990.

[11] R. Ovans and R. Davison, "An interactive constraint-based expert assistant for music composition," in *Proc. Ninth Canadian Conf. Artificial Intelligence*, pp. 76–81, 1996.

[12] S. Davismoon and J. Eccles, "Combining musical constraints with markov transition probabilities to improve the generation of creative musical structures," in *European Conference on the Applications of Evolutionary Computation*, pp. 361–370, Springer, 2010.

[13] F. Pachet and P. Roy, "Markov constraints : steerable generation of markov sequences," *Constraints*, vol. 16, no. 2, pp. 148–172, 2011.

[14] F. Pachet, P. Roy, G. Barbieri, and S. C. Paris, "Finite-length markov processes with constraints," *transition*, vol. 6, no. 1/3, 2011.

[15] F. Pachet and P. Roy, "Imitative leadsheet generation with user constraints," in *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, pp. 1077–1078, IOS Press, 2014.

[16] G. Boenn, M. Brain, M. De Vos, *et al.*, "Automatic composition of melodic and harmonic music by answer set programming," in *International Conference on Logic Programming*, pp. 160–174, Springer, 2008.

[17] D. Zimmermann, "Modelling musical structures," *Constraints*, vol. 6, no. 1, pp. 53–83, 2001.

[18] C. Truchet, "Omclouds, a library for musical constraints," in *Constraint Programming in Music* (C. Truchet and G. Assayag, eds.), ch. 8, pp. 189–213, ISTE-Wiley, 2011.

[19] C. Rueda, M. Lindberg, M. Laurson, G. Bloch, and G. Assayag, "Integrating constraint programming in visual musical composition languages," in *ECAI 98 Workshop on Constraints for Artistic Applications*, 1998.

[20] The MIDI Manufacturers Association, *The Complete MIDI 1.0 Detailed Specification*, 1996. document version 96.1 third edition.

[21] F. Pachet, "The muses system : an environment for experimenting with knowledge representation techniques in tonal harmony," in *First Brazilian Symposium on Computer Music, SBC&M*, vol. 94, pp. 3–4, Citeseer, 1994.

[22] P. Roy and F. Pachet, "Reifying constraint satisfaction in smalltalk," *JOOP*, vol. 10, no. 4, pp. 43–51, 1997.

[23] T. Anders, *Composing music by composing rules : Design and usage of a generic music constraint system*. PhD thesis, Queen's University Belfast, 2007.

[24] A. Papadopoulos, P. Roy, and F. Pachet, "Assisted lead sheet composition using flowcomposer," in *International Conference on Principles and Practice of Constraint Programming*, pp. 769–785, Springer, 2016.

[25] P. Roy and F. Pachet, "Enforcing meter in finite-length markov sequences.," in *AAAI*, 2013.

[26] A. Papadopoulos, F. Pachet, P. Roy, and J. Sakellariou, "Exact sampling for regular and markov constraints with belief propagation," in *International Conference on Principles and Practice of Constraint Programming*, pp. 341–350, Springer, 2015.

[27] P. Roy, G. Perez, J.-C. Régin, A. Papadopoulos, F. Pachet, and M. Marchini, "Enforcing structure on temporal sequences : the allen constraint," in *International conference on principles and practice of constraint programming*, pp. 786–801, Springer, 2016.

[28] B. Janssen, W. B. De Haas, A. Volk, and P. Van Kranenburg, "Finding repeated patterns in music : State of knowledge, challenges, perspectives," in *International Symposium on Computer Music Modeling and Retrieval*, pp. 277–297, Springer, 2013.

[29] I. Karydis, A. Nanopoulos, and Y. Manolopoulos, "Finding maximum-length repeating patterns in music databases," *Multimedia Tools and Applications*, vol. 32, no. 1, pp. 49–71, 2007.

[30] P.-Y. Rolland, "Discovering patterns in musical sequences," *Journal of New Music Research*, vol. 28, no. 4, pp. 334–350, 1999.

[31] P. Weiner, "Linear pattern matching algorithms," in *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pp. 1–11, IEEE, 1973.

[32] M. Jekovec, J. Demsar, and A. Brodnik, "Computer aided melodic analysis using suffix tree," in *ICMC*, 2012.

[33] E. Cambouropoulos, M. Crochemore, C. S. Iliopoulos, M. Mohamed, and M.-F. Sagot, "All maximal-pairs in step–leap representation of melodic sequence," *Information Sciences*, vol. 177, no. 9, pp. 1954–1962, 2007.

[34] I. Knopke and F. Jürgensen, "A system for identifying common melodic phrases in the masses of palestrina," *Journal of New Music Research*, vol. 38, no. 2, pp. 171–181, 2009.

[35] W. Lee and A. L. Chen, "Efficient multifeature index structures for music data retrieval," in *Storage and Retrieval for Media Databases 2000*, vol. 3972, pp. 177–189, International Society for Optics and Photonics, 1999.

[36] M.-K. Shan and S.-C. Chiu, "Algorithmic compositions based on discovered musical patterns," *Multimedia Tools and Applications*, vol. 46, no. 1, p. 1, 2010.

[37] D. Conklin, "Discovery of distinctive patterns in music," *Intelligent Data Analysis*, vol. 14, no. 5, pp. 547–554, 2010.

[38] D. Herremans and E. Chew, "Morpheus : automatic music generation with recurrent pattern constraints and tension profiles," in *IEEE TENCON*, 2016.

[39] M. Pearce and G. Wiggins, "Towards a framework for the evaluation of machine compositions," in *Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences*, pp. 22–32, 2001.

[40] C. Roig, L. J. Tardón, I. Barbancho, and A. M. Barbancho, "Automatic melody composition based on a probabilistic model of music style and harmonic rules," *Knowledge-Based Systems*, vol. 71, pp. 419–434, 2014.

[41] T. Collins, R. Laney, A. Willis, and P. H. Garthwaite, "Developing and evaluating computational models of musical style," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 30, no. 01, p. 16–43, 2015.

[42] S. Seymour, "Nottingham database." `https://ifdo.ca/~seymour/nottingham/nottingham.html`. Accessed : 2018-01-12.

[43] C. Walshaw, "The abc music standard 2.1 (dec 2011)." `http://abcnotation.com/wiki/abc:standard:v2.1`.

[44] M. Nelson, "Fast string searching with suffix trees." `http://marknelson.us/1996/08/01/suffix-trees/`. Accessed : 2018-01-25.

[45] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995.

ANNEXE A DÉTAILS D'UTILISATION DU SYSTÈME

A.1 Suite d'accords

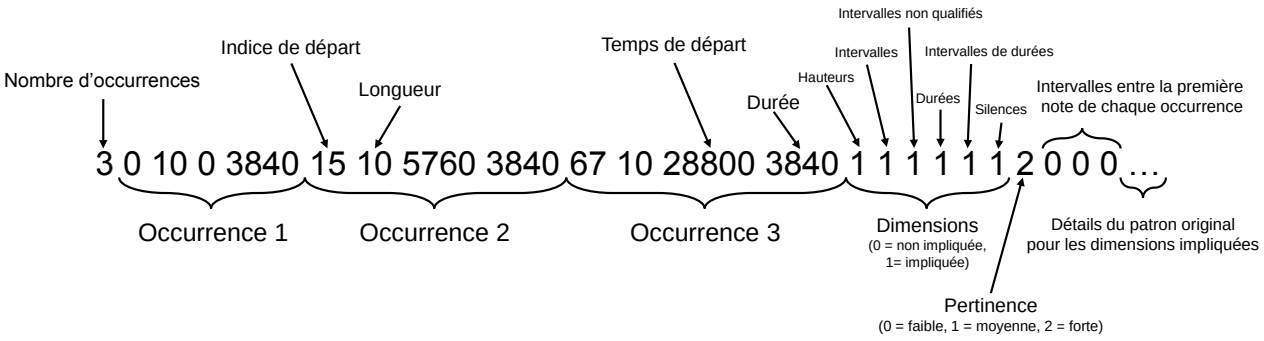
Les accords doivent être dans un fichier texte fourni à l'exécutable avec l'option `-file`. La suite d'accords doit être organisée de la façon suivante : `Tonalité Fondamentale Qualification Durée Fondamentale Qualification Durée ...`. La fondamentale doit être spécifiée avec la notation anglo-saxonne (lettres de A à G) et la qualification doit figurer parmi `maj, min, maj7, min7, aug, dim`. La durée des accords est exprimée en termes de noires.

A.2 Patrons rythmiques

La liste des patrons rythmiques est placée directement après la suite d'accords, dans le même fichier, sur la ligne qui suit. Les durées sont indiquées en termes de noires et séparées par un espace. Le nombre d'occurrences de chaque patron est indiqué après celui-ci avec deux points. Par exemple, si le patron noire noire noire est répété cinq fois : `1 1 1 : 5`. Il y a un patron par ligne.

A.3 Patrons structuraux

La liste des patrons structuraux fournis par l'algorithme doit être placée dans un fichier nommé `patterns.txt`. L'algorithme décrit les patrons de la manière suivante :



L'utilisateur est libre de modifier les informations des patrons à sa guise tant qu'il respecte ce format, qui est reconnu par le système.

A.4 Contraintes utilisateur

Dans la version actuelle du système, les contraintes doivent figurer dans un fichier texte qui doit être nommé `constraints.txt` (macro configurable dans `ConstraintParser.h`). La première ligne du fichier permet à l'utilisateur de fixer quatre paramètres. La première lettre détermine si les patrons rythmiques proviennent de la valse d'origine (W) ou du corpus (C). Les trois lettres suivantes précisent le niveau de pertinence appliqué aux patrons forts (S), moyens (A) et faibles (W), dans cet ordre. L'utilisateur peut donc par exemple faire en sorte que les patrons qui étaient fortement pertinents dans la valse d'origine ne soient que faiblement pertinents dans le résultat généré. Il est également possible de désactiver un groupe de patrons (0). Le fichier doit être organisé de la façon suivante :

W/C S/A/W/0 S/A/W/0 S/A/W/0
phase1
`contrainte1`
`contrainte2`
`contrainte3`
...
phase2
`contrainte1`
`contrainte2`
`contrainte3`
...

Les contraintes doivent être sous la forme `variables,contrainte,variables/paramètre` (comme les exemples de la section A.4.1).

A.4.1 Langage de déclaration des contraintes utilisateur

Pour que l'utilisateur soit capable de déclarer ses contraintes pour les deux phases avant de lancer le programme, il a fallu mettre en place un langage simple. Cela permet à l'utilisateur de déclarer ses contraintes de phase 1 et de phase 2 dans un fichier texte qui sera interprété en temps voulu par le programme.

A.4.2 Accès aux variables

Les contraintes de phase 1 peuvent agir sur trois propriétés d'une mesure : le nombre de notes, l'indice du patron rythmique et les durées des notes. Pour indiquer sur quelles variables poser les contraintes, il faut spécifier la position de la mesure avec la lettre `b` (pour *bar* en anglais) : pour la première mesure on écrira `b1`, pour les mesures trois à cinq, on écrira `b3-5`.

L'utilisateur doit aussi indiquer sur quelle dimension appliquer sa contrainte : `.p` pour patron rythmique, `.d` pour durée et aucune lettre pour le nombre de notes. Il peut ensuite spécifier ou non des variables en particulier dans la mesure choisie. Exemples :

`b1.p` accède au patron rythmique de la première mesure

`b7.d` pour les durées de la mesure numéro 7

`b1.d3` pour la durée de la troisième note de la première mesure

`b12.d1-3` pour les durées des notes 1 à 3 de la mesure 12

`b2` pour le nombre de notes de la deuxième mesure

Pour la phase 2, il faut indiquer qu'on traite les hauteurs avec la lettre `h`. On peut accéder aux variables de la même façon que pour la première phase. Exemples :

`b1.h` accède aux hauteurs de la première mesure

`b1.h3` pour la hauteur de la troisième note de la première mesure

`b12.h1-3` pour les hauteurs des notes 1 à 3 de la mesure 12

A.4.3 Domaines des variables

Pour la première phase, les patrons rythmiques sont identifiés par leur indice dans la liste des patrons fournie au système. Les durées quant à elles doivent être indiquées au format MIDI, une noire correspondant à la valeur 480.

Dans la seconde phase, les hauteurs sont référencées par leur degré sur l'échelle chromatique par rapport à la tonalité du morceau, de 0 à 11.

A.4.4 Contraintes classiques

Les opérateurs $=, !=, <, >$ ainsi que la contrainte globale ALLDIFFERENT sont disponibles. Il est donc possible de poser des contraintes comme `b3,=,b4` en phase 1 pour indiquer que les mesures 3 et 4 doivent avoir le même nombre de notes, ou encore `b1.d,>,120` pour faire en sorte que les durées des notes du bloc 1 soient toutes supérieures à une double croche.

Il est également possible d'utiliser les opérateurs =+ et =* pour effectuer des transpositions sur les hauteurs et sur les durées respectivement. Par exemple, pour indiquer que les hauteurs des notes de la mesure 4 doivent être une transposition d'un ton vers le haut (différence de +2 en MIDI) des notes du bloc 2 on écrit : `b4.h,=+2,b2.h` (en phase 2 uniquement).

La contrainte globale `AllDifferent` s'utilise simplement de la façon suivante : `b6.h1-3,alldiff` pour indiquer que les hauteurs des notes 1 à 3 de la mesure 6 doivent toutes être différentes.

A.4.5 Contraintes de cardinalité

Les contraintes de cardinalité sont exprimées en termes de pourcentages et prennent donc un paramètre supplémentaire. Les opérateurs sont : %=, %!=, %< et %>. Ainsi, pour imposer que plus de trente pour cent des notes d'une mélodie de 25 mesures soient des croches, on écrit : `b1-25.d,%>,30,240`. L'utilisateur peut également influencer la proportion des patrons rythmiques les plus fréquents de la valse d'origine à l'aide de la lettre `f` (pour fréquence). Par exemple : `b1-25.f,%<,20,3` signifie : les mesures 1 à 25 doivent contenir moins de 20% de patrons rythmiques appartenant aux 3 plus fréquents de la valse d'origine. Ce type de contraintes permet notamment de contrôler la proportion entre patrons de la valse et patrons du corpus lorsque l'utilisateur choisit d'utiliser ces derniers.

A.4.6 Contraintes implicites

Des contraintes implicites sont interprétées automatiquement par le système en fonction des contraintes de phase 2 spécifiées par l'utilisateur. Cela permet au modèle de rester cohérent. Par exemple, si la contrainte `b3.h,=,b6.h` est posée en phase 2, le système va automatiquement faire en sorte que les mesures 3 et 6 aient le même nombre de notes. De même, si l'utilisateur cherche à influencer les durées des notes 2 à 5 de la mesure 2 par exemple, le système va s'assurer que cette dernière ait au moins 5 notes.

ANNEXE B EXEMPLES DE GÉNÉRATION

B.1 Patrons rythmiques du corpus

```
0.5 0.5 0.5 0.5 0.5 0.5 : 113
0.5 0.5 0.5 0.5 1 : 13
0.5 0.5 0.5 1 0.5 : 1
0.5 0.5 1 0.5 0.5 : 15
0.5 0.5 1 1 : 25
0.5 0.5 1.5 0.5 : 5
0.5 0.5 2 : 9
0.5 1 1 0.5 : 2
0.5 1 1.5 : 3
0.5 1.5 0.5 0.5 : 1
0.5 1.5 1 : 14
1 0.5 0.5 0.5 0.5 : 150
1 0.5 0.5 1 : 30
1 0.5 1 0.5 : 6
1 0.5 1.5 : 1
1 1 0.5 0.5 : 54
1 1 1 : 743
1 1.5 0.5 : 44
1 2 : 38
1.5 0.5 0.5 0.5 : 107
1.5 0.5 1 : 253
1.5 1 0.5 : 1
2 0.5 0.5 : 189
2 1 : 505
3 : 206
```

B.2 Exemple 4.1

B.2.1 Accords et patrons rythmiques

```
D maj D maj 1 D maj 3 D maj 3 E min 3 A 7 3 D maj 3 D maj 3 G maj 2 A 7 1
D maj 3 B min 3 F# min 3 B min 3 F# min 3 B min 3 A maj 3 D maj 2 A maj
1 D maj 2 A maj 1 B min 3 F# min 3 B min 2 E 7 1 A 7 3 D maj 3 D maj 3 G
maj 2 A 7 1 D maj 2
0.5 0.5 0.5 0.5 0.5 0.5 : 0
0.5 0.5 0.5 0.5 1 : 0
0.5 0.5 0.5 1 0.5 : 0
0.5 0.5 1 0.5 0.5 : 0
0.5 0.5 1 1 : 0
0.5 0.5 1.5 0.5 : 0
0.5 0.5 2 : 0
0.5 1 1 0.5 : 0
0.5 1 1.5 : 0
0.5 1.5 0.5 0.5 : 0
0.5 1.5 1 : 0
1 0.5 0.5 0.5 0.5 : 5
1 0.5 0.5 1 : 0
1 0.5 1 0.5 : 0
1 0.5 1.5 : 0
1 1 0.5 0.5 : 0
1 1 1 : 5
1 1.5 0.5 : 0
1 2 : 0
1.5 0.5 0.5 0.5 : 5
1.5 0.5 1 : 3
1.5 1 0.5 : 0
2 0.5 0.5 : 0
2 1 : 5
3 : 0
```

B.2.2 Patrons structurels

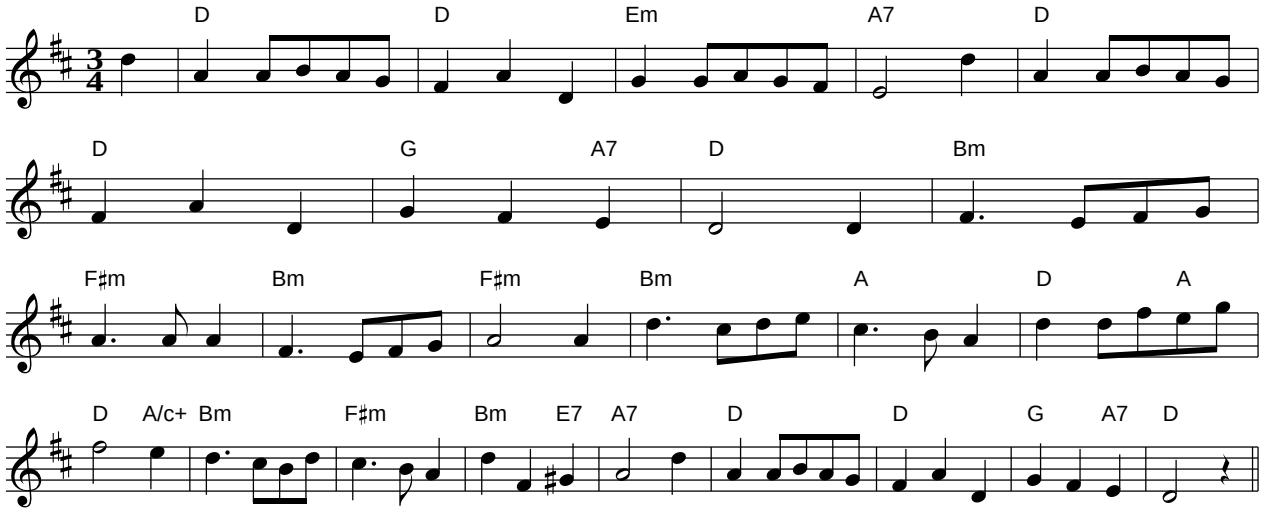
```
3 0 10 0 3840 15 10 5760 3840 67 10 28800 3840 1 1 1 1 1 1 2 0 0 0 74
69 69 71 69 67 66 69 62 67 -5 0 2 -2 -2 -1 3 -7 5 -3 0 1 -1 -1 -1 2 -4 3
480 480 240 240 240 240 480 480 480 480 100000 50000 100000 100000 100000
200000 100000 100000 100000 0 0 0 0 0 0 0 0 0 0
2 15 13 5760 5760 67 13 28800 5760 1 1 1 1 1 1 2 0 0 74 69 69 71 69 67 66
69 62 67 66 64 62 -5 0 2 -2 -2 -1 3 -7 5 -1 -2 -2 -3 0 1 -1 -1 -1 2 -4 3
-1 -1 -1 480 480 240 240 240 240 480 480 480 480 480 480 960 100000 50000
100000 100000 100000 200000 100000 100000 100000 100000 100000 200000 0 0
0 0 0 0 0 0 0 0 0 0
2 29 4 12000 1440 36 4 14880 1440 1 1 1 1 1 1 2 0 0 66 64 66 67 -2 2 1 -1
1 1 720 240 240 240 33333 100000 100000 0 0 0 0
2 46 4 19200 1920 60 4 24960 1920 1 1 1 1 1 1 2 0 0 73 71 69 74 -2 -2 5
-1 -1 3 720 240 480 480 33333 200000 100000 0 0 0 0
4 1 5 480 1440 9 5 3360 1440 16 5 6240 1440 68 5 29280 1440 0 0 1 1 1 1 1
0 2 0 0 0 1 -1 -1 480 240 240 240 240 50000 100000 100000 100000 0 0 0 0
0
2 27 2 10560 1440 40 2 16320 1440 0 1 1 1 1 1 1 0 -7 0 0 960 480 50000 0
0
3 29 4 12000 1440 36 4 14880 1440 42 4 17760 1440 0 0 1 1 1 1 1 0 0 -8 -1
1 1 720 240 240 240 33333 100000 100000 0 0 0 0
5 0 6 0 1920 8 6 2880 1920 15 6 5760 1920 48 6 20160 1920 67 6 28800 1920
0 0 0 1 1 1 0 0 12 0 5 0 480 480 240 240 240 240 100000 50000 100000
100000 100000 0 0 0 0 0 0
12 6 3 1920 1440 7 3 2400 1440 21 3 7680 1440 22 3 8160 1440 23 3 8640
1440 24 3 9120 1440 62 3 25920 1440 63 3 26400 1440 73 3 30720 1440 74 3
31200 1440 75 3 31680 1440 76 3 32160 1440 0 0 0 1 1 1 0 0 -3 0 -3 4 -1
-3 -8 0 -3 4 -1 480 480 480 100000 100000 0 0 0
2 8 8 2880 3360 48 8 20160 3360 0 0 0 1 1 1 0 0 -7 480 480 240 240 240
240 960 480 100000 50000 100000 100000 100000 400000 50000 0 0 0 0 0 0 0
0
3 11 5 4080 2160 37 5 15600 2160 51 5 21360 2160 0 0 0 1 1 1 0 0 5 -9 240
240 240 960 480 100000 100000 400000 50000 0 0 0 0 0
2 14 14 4800 6720 66 14 27840 6720 0 0 0 1 1 1 0 0 -5 960 480 480 240
240 240 240 480 480 480 480 480 480 960 50000 100000 50000 100000 100000
```

```
100000 200000 100000 100000 100000 100000 100000 200000 0 0 0 0 0 0 0 0 0
0 0 0 0 0
5 14 2 4800 1440 27 2 10560 1440 40 2 16320 1440 54 2 22080 1440 66 2
27840 1440 0 0 0 1 1 1 0 0 2 -5 -14 -5 960 480 50000 0 0
4 21 5 7680 2400 22 5 8160 2400 73 5 30720 2400 74 5 31200 2400 0 0 0 1 1
1 0 0 -3 0 -3 480 480 480 480 480 100000 100000 100000 100000 0 0 0 0 0
2 23 6 8640 3360 62 6 25920 3360 0 0 0 1 1 1 0 0 -7 480 480 480 480 960
480 100000 100000 100000 200000 50000 0 0 0 0 0 0
3 23 5 8640 2880 62 5 25920 2880 75 5 31680 2880 0 0 0 1 1 1 0 0 -7 0 480
480 480 480 960 100000 100000 100000 200000 0 0 0 0 0
3 27 9 10560 4320 40 9 16320 4320 54 9 22080 4320 0 0 0 1 1 1 0 0 -7 -16
960 480 720 240 240 240 720 240 480 50000 150000 33333 100000 100000
300000 33333 200000 0 0 0 0 0 0 0 0 0
4 28 5 11520 1920 35 5 14400 1920 41 5 17280 1920 55 5 23040 1920 0 0 0 1
1 1 0 0 -7 -7 -14 480 720 240 240 240 150000 33333 100000 100000 0 0 0 0
0
2 37 13 15600 5520 51 13 21360 5520 0 0 0 1 1 1 0 0 -14 240 240 240 960
480 720 240 240 240 720 240 480 480 100000 100000 400000 50000 150000
33333 100000 100000 300000 33333 200000 100000 0 0 0 0 0 0 0 0 0 0 0 0 0
```

B.2.3 Contraintes utilisateur

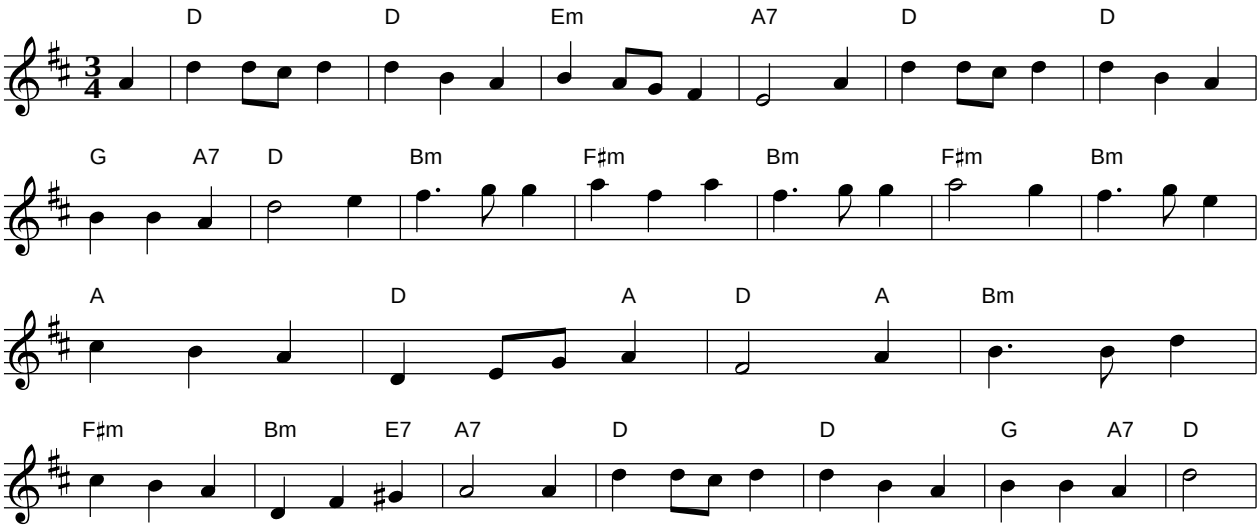
```
C S W W
phase1
b2.p,!=,0
b2.p,!=,11
phase2
```

B.2.4 Valse d'origine




```
X:9
T:Caerdroea
S:JP/AF 85
M:3/4
L:1/4
K:D
d|"D"AA/2B/2A/2G/2|"D"FAD|"Em"GG/2A/2G/2F/2|"A7"E2d|"D"AA/2B/2A/2G/2|\
"D"FAD| "G"GF"A7"E|"D"D2:|
D|"Bm"F3/2E/2F/2G/2|"F#m"A3/2A/2A|"Bm"F3/2E/2F/2G/2|"F#m"A2A|\
"Bm"d3/2c/2d/2e/2|"A"c3/2B/2A|"D"dd/2f/2"A"e/2g/2|
"D"f2"A/c+"e|"Bm"d3/2c/2B/2d/2|"F#m"c3/2B/2A|"Bm"dF"E7"Ĝ|"A7"A2d|\
"D"AA/2B/2A/2G/2|"D"FAD|"G"GF"A7"E|"D"D2||
```

B.2.5 Valse générée




```
X:1
R:Waltz
S:melody generation constraint system
M:3/4
L:1/4
K:D
A|"D"dd/2c/2d|"D"dBA|"Em"BA/2G/2F|"A7"E2A|"D"dd/2c/2d|"D"dBA|"G"BB"A7"A|\
"D"d2e|"Bm"f3/2g/2g|"F#m"afa|"Bm"f3/2g/2g|"F#m"a2g|"Bm"f3/2g/2e|"A"cBA|\
"D"DE/2G/2"A"A|"D"F2"A"A|"Bm"B3/2B/2d|"F#m"cBA|"Bm"DF"E7"^G|\
"A7"A2A|"D"dd/2c/2d|"D"dBA|"G"BB"A7"A|"D"d2|
```

B.3 Exemple 4.2

B.3.1 Accords et patrons rythmiques

```
F maj F maj 1 F maj 3 F maj 3 Bb maj 3 F maj 2 C 7 1 F maj 3 F maj 3 G
min 3 C 7 3 F maj 3 F maj 3 Bb maj 3 F maj 2 C 7 1 F maj 3 C 7 3 F maj 2
Bb maj 1 F maj 3 F maj 3 F maj 3 C 7 3 C 7 3 C 7 3 C 7 3 F maj 3 F maj 3
F maj 3 C 7 3 F maj 3 Bb maj 3 F maj 3 C 7 3 F maj 2 Bb maj 1 F maj 2
0.5 0.5 0.5 0.5 0.5 0.5 : 0
0.5 0.5 0.5 0.5 1 : 0
```

```
0.5 0.5 0.5 1 0.5 : 0
0.5 0.5 1 0.5 0.5 : 0
0.5 0.5 1 1 : 0
0.5 0.5 1.5 0.5 : 0
0.5 0.5 2 : 0
0.5 1 1 0.5 : 0
0.5 1.5 0.5 0.5 : 0
0.5 1.5 1 : 6
1 0.5 0.5 0.5 0.5 : 0
1 0.5 0.5 1 : 0
1 0.5 1 0.5 : 0
1 0.5 1.5 : 0
1 1 0.5 0.5 : 0
1 1 1 : 8
1 1.5 0.5 : 0
1 2 : 0
1.5 0.5 0.5 0.5 : 0
1.5 0.5 1 : 12
1.5 1 0.5 : 0
2 0.5 0.5 : 1
2 1 : 4
3 : 0
```

B.3.2 Patrons structurels

```
2 2 14 480 7200 25 14 12000 7200 1 1 1 1 1 1 2 0 0 72 74 72 72 74 72 77
76 74 72 70 69 72 65 2 -2 0 2 -2 5 -1 -2 -2 -2 -1 3 -7 1 -1 0 1 -1 3 -1
-1 -1 -1 -1 2 -4 720 240 480 720 240 480 480 480 480 960 480 240 720 480
33333 200000 150000 33333 200000 100000 100000 100000 200000 50000 50000
300000 66666 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 2 3 480 1440 5 3 1920 1440 25 3 12000 1440 28 3 13440 1440 1 1 1 1 1 1
2 0 0 0 0 72 74 72 2 -2 1 -1 720 240 480 33333 200000 0 0 0
4 13 3 6240 1440 16 3 7680 1440 36 3 17760 1440 82 3 40800 1440 1 1 1 1 1
1 2 0 0 0 0 69 72 65 3 -7 2 -4 240 720 480 300000 66666 0 0 0
2 21 4 10080 1920 60 4 30240 1920 1 1 1 1 1 1 2 0 0 67 67 69 70 0 2 1 0 1
```

```
1 480 480 480 480 100000 100000 100000 0 0 0 0
2 35 11 17280 5760 81 11 40320 5760 1 1 1 1 1 1 2 0 0 70 69 72 65 67 72
60 65 67 65 65 -1 3 -7 2 5 -12 5 2 -2 0 -1 2 -4 1 3 -7 3 1 -1 0 480 240
720 480 240 720 480 720 240 480 960 50000 300000 66666 50000 300000 66666
150000 33333 200000 200000 0 0 0 0 0 0 0 0 0 0 0
7 2 3 480 1440 5 3 1920 1440 19 3 9120 1440 25 3 12000 1440 28 3 13440
1440 42 3 20640 1440 88 3 43680 1440 0 0 1 1 1 1 1 0 0 5 0 0 7 7 1 -1 720
240 480 33333 200000 0 0 0
3 7 3 2880 1440 30 3 14400 1440 50 3 24960 1440 0 0 1 1 1 1 1 0 0 7 3 -1
480 480 480 100000 100000 0 0 0
3 8 3 3360 1440 31 3 14880 1440 79 3 39360 1440 0 0 1 1 1 1 1 0 0 3 -1 -1
480 480 480 100000 100000 0 0 0
2 18 3 8640 1440 24 3 11520 1440 0 0 1 1 1 1 1 0 -5 1 1 480 720 240
150000 33333 0 0 0
2 47 4 23520 1920 58 4 29280 1920 0 0 1 1 1 1 1 0 -2 0 0 0 720 240 480
480 33333 200000 100000 0 0 0 0
2 54 3 26880 1920 65 3 32640 1920 0 0 1 1 1 1 1 0 -9 0 0 480 480 960
100000 200000 0 0 0
3 70 3 35040 1440 73 3 36480 1440 76 3 37920 1440 0 0 1 1 1 1 1 0 -1 -3
-1 1 720 240 480 33333 200000 0 0 0
2 70 6 35040 2880 73 6 36480 2880 0 0 1 1 1 1 1 0 -1 -1 1 1 -1 1 720 240
480 720 240 480 33333 200000 150000 33333 200000 0 0 0 0 0 0
2 0 8 0 3360 68 8 34560 3360 0 0 0 1 1 1 0 0 4 240 240 720 240 480 720
240 480 100000 300000 33333 200000 150000 33333 200000 0 0 0 0 0 0 0 0
2 2 20 480 10080 25 20 12000 10080 0 0 0 1 1 1 0 0 0 720 240 480 720 240
480 480 480 480 960 480 240 720 480 240 720 480 720 240 480 33333 200000
150000 33333 200000 100000 100000 100000 200000 50000 50000 300000 66666
50000 300000 66666 150000 33333 200000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
12 2 3 480 1440 5 3 1920 1440 19 3 9120 1440 25 3 12000 1440 28 3 13440
1440 42 3 20640 1440 47 3 23520 1440 58 3 29280 1440 70 3 35040 1440 73 3
36480 1440 76 3 37920 1440 88 3 43680 1440 0 0 0 1 1 1 0 0 0 5 0 0 7 7 5
3 2 0 7 720 240 480 33333 200000 0 0 0
3 2 9 480 4320 25 9 12000 4320 73 9 36480 4320 0 0 0 1 1 1 0 0 0 2 720
240 480 720 240 480 480 480 480 33333 200000 150000 33333 200000 100000
```

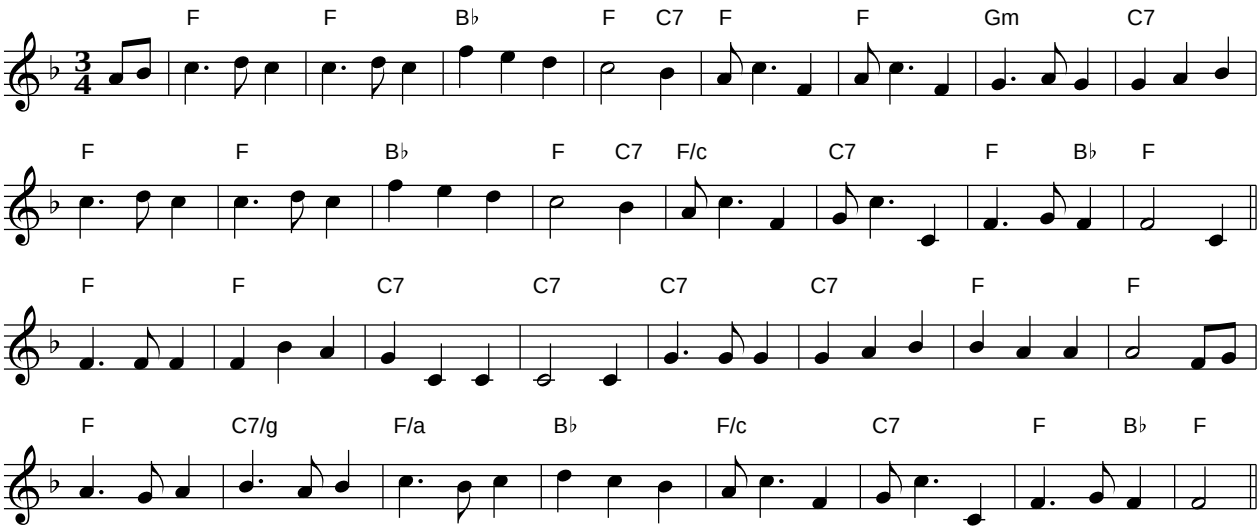
```
100000 100000 0 0 0 0 0 0 0 0 0
4 2 6 480 2880 25 6 12000 2880 70 6 35040 2880 73 6 36480 2880 0 0 0 1 1
1 0 0 0 3 2 720 240 480 720 240 480 33333 200000 150000 33333 200000 0 0
0 0 0 0
6 4 7 1440 3360 18 7 8640 3360 27 7 12960 3360 46 7 23040 3360 57 7 28800
3360 75 7 37440 3360 0 0 0 1 1 1 0 0 7 0 12 12 2 480 720 240 480 480 480
480 150000 33333 200000 100000 100000 100000 0 0 0 0 0 0 0
10 4 4 1440 1920 18 4 8640 1920 24 4 11520 1920 27 4 12960 1920 41 4
20160 1920 46 4 23040 1920 57 4 28800 1920 72 4 36000 1920 75 4 37440
1920 87 4 43200 1920 0 0 0 1 1 1 0 0 7 2 0 12 12 12 3 2 12 480 720 240
480 150000 33333 200000 0 0 0 0
3 7 6 2880 3360 30 6 14400 3360 52 6 25920 3360 0 0 0 1 1 1 0 0 0 3 480
480 480 480 960 480 100000 100000 100000 200000 50000 0 0 0 0 0 0
18 7 3 2880 1440 8 3 3360 1440 21 3 10080 1440 22 3 10560 1440 30 3 14400
1440 31 3 14880 1440 49 3 24480 1440 50 3 24960 1440 51 3 25440 1440 52
3 25920 1440 53 3 26400 1440 60 3 30240 1440 61 3 30720 1440 62 3 31200
1440 63 3 31680 1440 64 3 32160 1440 78 3 38880 1440 79 3 39360 1440 0
0 0 1 1 1 0 0 -5 5 5 0 -5 7 7 2 3 5 5 5 3 2 2 0 -2 480 480 480 100000
100000 0 0 0
4 7 5 2880 2880 30 5 14400 2880 52 5 25920 2880 63 5 31680 2880 0 0 0 1 1
1 0 0 0 3 2 480 480 480 480 960 100000 100000 100000 200000 0 0 0 0 0
4 10 3 4320 1920 33 3 15840 1920 44 3 21600 1920 55 3 27360 1920 0 0 0 1
1 1 0 0 0 9 14 480 960 480 200000 50000 0 0 0
6 12 4 5760 1920 15 4 7200 1920 35 4 17280 1920 38 4 18720 1920 81 4
40320 1920 84 4 41760 1920 0 0 0 1 1 1 0 0 5 0 5 0 5 480 240 720 480
50000 300000 66666 0 0 0 0
2 24 10 11520 4800 72 10 36000 4800 0 0 0 1 1 1 0 0 1 480 720 240 480 720
240 480 480 480 480 150000 33333 200000 150000 33333 200000 100000 100000
100000 0 0 0 0 0 0 0 0 0 0
2 44 13 21600 7200 55 13 27360 7200 0 0 0 1 1 1 0 0 5 480 960 480 720 240
480 480 480 480 480 480 480 960 200000 50000 150000 33333 200000 100000
100000 100000 100000 100000 100000 200000 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6 49 5 24480 2400 50 5 24960 2400 51 5 25440 2400 60 5 30240 2400 61 5
30720 2400 62 5 31200 2400 0 0 0 1 1 1 0 0 0 -5 -2 -2 -4 480 480 480 480
480 100000 100000 100000 100000 0 0 0 0 0
```

```
4 49 6 24480 2880 50 6 24960 2880 60 6 30240 2880 61 6 30720 2880 0 0 0
1 1 1 0 0 0 -2 -2 480 480 480 480 480 480 100000 100000 100000 100000
100000 0 0 0 0 0 0
```

B.3.3 Contraintes utilisateur

```
C S W W
phase1
phase2
```

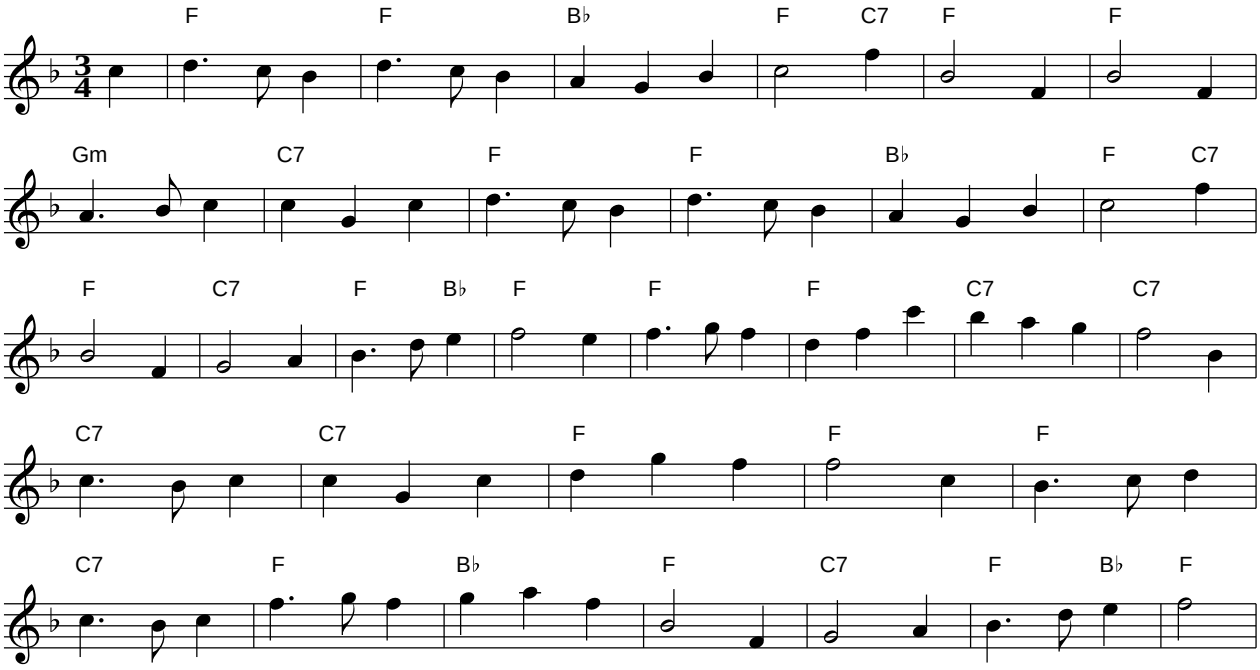
B.3.4 Valse d'origine




```
X:8
T:Bonnie Dundee
S:Joy Foxley
M:3/4
L:1/4
K:F
A/2B/2|"F"c3/2d/2c|"F"c3/2d/2c|"Bb"fed|"F"c2"C7"B|"F"A/2c3/2F|"F"A/2c3/2F|\
"Gm"G3/2A/2G|"C7"GAB|
"F"c3/2d/2c|"F"c3/2d/2c|"Bb"fed|"F"c2"C7"B|"F/c"A/2c3/2F|"C7"G/2c3/2C|\
"F"F3/2G/2"Bb"F|"F"F2C||
```

"F"F3/2F/2F|"F"FBA|"C7"GCC|"C7"C2C|"C7"G3/2G/2G|"C7"GAB|"F"BAA|\
"F"A2F/2G/2|"F"A3/2G/2A|"C7/g"B3/2A/2B|"F/a"c3/2B/2c|"Bb"dcB|"F/c"A/2c3/2F|\
"C7"G/2c3/2C|"F"F3/2G/2"Bb"F|"F"F2||

B.3.5 Valse générée



X:1
T:Original waltz
R:Waltz
S:melody generation constraint system
M:3/4
L:1/4
K:F
c|"F"d3/2c/2B|"F"d3/2c/2B|"Bb"AGB|"F"c2"C7"f|"F"B2F|"F"B2F|"Gm"A3/2B/2c|"C7"c
Gc|"F"d3/2c/2B|"F"d3/2c/2B|"Bb"AGB|"F"c2"C7"f|"F"B2F|"C7"G2A|"F"B3/2d/2"Bb"e|
"F"f2e|"F"f3/2g/2f|"F"dfc'|"C7"bag|"C7"f2B|"C7"c3/2B/2c|"C7"cGc|"F"dgf|"F"f2c|"F"B3/
2c/2d|"C7"c3/2B/2c|"F"f3/2g/2f|"Bb"gaf|"F"B2F|"C7"G2A|"F"B3/2d/2"Bb"e|"F"f2|

B.4 Exemple 4.3

B.4.1 Accords et patrons rythmiques

```
G maj G maj 1 G maj 3 G maj 3 C maj 3 D 7 3 G maj 3 G maj 3 C maj 3 D 7 3
G maj 3 G maj 3 C maj 3 D 7 3 G maj 3 G maj 3 C maj 2 D 7 1 G maj 3 D maj
3 D maj 3 C maj 3 G maj 3 D maj 3 D maj 3 C maj 3 D 7 3 G maj 3 G maj 3 C
maj 3 D 7 3 G maj 3 G maj 3 C maj 2 D 7 1 G maj 2
0.5 0.5 0.5 0.5 0.5 0.5 : 0
0.5 0.5 0.5 0.5 1 : 0
0.5 0.5 0.5 1 0.5 : 0
0.5 0.5 1 0.5 0.5 : 0
0.5 0.5 1 1 : 0
0.5 0.5 1.5 0.5 : 0
0.5 0.5 2 : 0
0.5 1 1 0.5 : 0
0.5 1 1.5 : 0
0.5 1.5 0.5 0.5 : 0
0.5 1.5 1 : 0
1 0.5 0.5 0.5 0.5 : 1
1 0.5 0.5 1 : 0
1 0.5 1 0.5 : 0
1 0.5 1.5 : 0
1 1 0.5 0.5 : 3
1 1 1 : 17
1 1.5 0.5 : 5
1 2 : 0
1.5 0.5 0.5 0.5 : 1
1.5 0.5 1 : 0
1.5 1 0.5 : 0
2 0.5 0.5 : 3
2 1 : 1
3 : 0
```

B.4.2 Patrons structurels

```
3 0 23 0 10080 27 23 11520 10080 76 23 34560 10080 1 1 1 1 1 1 2 0 0 0 71
69 67 67 67 69 71 74 74 76 79 79 81 79 81 83 83 81 79 74 74 76 79 -2 -2
0 0 2 2 3 0 2 3 0 2 -2 2 2 0 -2 -2 -5 0 2 3 -1 -1 0 0 1 1 2 0 1 2 0 1 -1
1 1 0 -1 -1 -3 0 1 2 240 240 480 480 240 240 480 480 480 480 480 480 960
240 240 480 720 240 480 480 480 480 480 100000 200000 100000 50000 100000
200000 100000 100000 100000 100000 100000 200000 25000 100000 200000
150000 33333 200000 100000 100000 100000 100000 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
4 7 5 2400 2400 19 5 8160 2400 34 5 13920 2400 83 5 36960 2400 1 1 1 1
1 1 2 0 0 0 0 74 74 76 79 79 0 2 3 0 0 1 2 0 480 480 480 480 480 100000
100000 100000 100000 0 0 0 0 0
6 7 4 2400 1920 19 4 8160 1920 34 4 13920 1920 46 4 19680 1920 83 4 36960
1920 95 4 42720 1920 1 1 1 1 1 1 2 0 0 0 0 0 0 74 74 76 79 0 2 3 0 1 2
480 480 480 480 100000 100000 100000 0 0 0 0
2 26 25 11280 10800 75 25 34320 10800 1 1 1 1 1 1 2 0 0 72 71 69 67 67 67
69 71 74 74 76 79 79 81 79 81 83 83 81 79 74 74 76 79 78 -1 -2 -2 0 0 2
2 3 0 2 3 0 2 -2 2 2 0 -2 -2 -5 0 2 3 -1 -1 -1 -1 0 0 1 1 2 0 1 2 0 1 -1
1 1 0 -1 -1 -3 0 1 2 -1 240 240 240 480 480 240 240 480 480 480 480 480
480 960 240 240 480 720 240 480 480 480 480 480 480 100000 100000 200000
100000 50000 100000 200000 100000 100000 100000 100000 100000 200000
25000 100000 200000 150000 33333 200000 100000 100000 100000 100000
100000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 49 3 21120 1440 51 3 22080 1440 1 1 1 1 1 1 2 0 0 79 78 79 -1 1 -1 1
480 480 480 100000 100000 0 0 0
2 54 9 23520 4320 65 9 29280 4320 1 1 1 1 1 1 2 0 0 81 81 79 78 74 74 79
79 76 0 -2 -1 -4 0 5 0 -3 0 -1 -1 -2 0 3 0 -2 480 720 240 480 480 480 480
480 480 150000 33333 200000 100000 100000 100000 100000 100000 0 0 0 0 0
0 0 0 0
7 6 3 1920 1440 9 3 3360 1440 21 3 9120 1440 33 3 13440 1440 36 3 14880
1440 82 3 36480 1440 85 3 37920 1440 0 0 1 1 1 1 1 0 -5 -5 0 -5 0 -5 2 0
480 480 480 100000 100000 0 0 0
5 15 4 6240 1920 42 4 17760 1920 54 4 23520 1920 65 4 29280 1920 91 4
40800 1920 0 0 1 1 1 1 1 0 0 2 2 0 0 -1 -1 480 720 240 480 150000 33333
200000 0 0 0 0
```

```
6 0 4 0 1440 4 4 1440 1440 27 4 11520 1440 31 4 12960 1440 76 4 34560
1440 80 4 36000 1440 0 0 0 1 1 1 0 0 4 0 4 0 4 240 240 480 480 100000
200000 100000 0 0 0 0
3 0 24 0 10560 27 24 11520 10560 76 24 34560 10560 0 0 0 1 1 1 0 0 0
0 240 240 480 480 240 240 480 480 480 480 480 480 960 240 240 480 720
240 480 480 480 480 480 480 100000 200000 100000 50000 100000 200000
100000 100000 100000 100000 100000 200000 25000 100000 200000 150000
33333 200000 100000 100000 100000 100000 100000 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
2 0 25 0 11040 27 25 11520 11040 0 0 0 1 1 1 0 0 0 240 240 480 480 240
240 480 480 480 480 480 480 960 240 240 480 720 240 480 480 480 480 480
480 480 100000 200000 100000 50000 100000 200000 100000 100000 100000
100000 100000 200000 25000 100000 200000 150000 33333 200000 100000
100000 100000 100000 100000 100000 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0
4 2 4 480 1440 23 4 10080 1440 29 4 12000 1440 78 4 35040 1440 0 0 0 1 1
1 0 0 -12 0 0 480 480 240 240 100000 50000 100000 0 0 0 0
5 5 8 1680 4080 32 8 13200 4080 56 8 24720 4080 81 8 36240 4080 93 8
42000 4080 0 0 0 1 1 1 0 0 0 -10 0 -12 240 480 480 480 480 480 480 960
200000 100000 100000 100000 100000 100000 200000 0 0 0 0 0 0 0 0
8 5 7 1680 3120 17 7 7440 3120 32 7 13200 3120 44 7 18960 3120 56 7 24720
3120 67 7 30480 3120 81 7 36240 3120 93 7 42000 3120 0 0 0 1 1 1 0 0 -12
0 -12 -10 -10 0 -12 240 480 480 480 480 480 480 200000 100000 100000
100000 100000 100000 0 0 0 0 0 0 0
13 6 6 1920 2880 18 6 7680 2880 19 6 8160 2880 33 6 13440 2880 45 6 19200
2880 46 6 19680 2880 47 6 20160 2880 48 6 20640 2880 49 6 21120 2880 57 6
24960 2880 68 6 30720 2880 82 6 36480 2880 94 6 42240 2880 0 0 0 1 1 1 0
0 -8 -3 0 -8 -3 -3 -5 -8 -7 -7 0 -8 480 480 480 480 480 480 100000 100000
100000 100000 100000 0 0 0 0 0 0
37 6 3 1920 1440 7 3 2400 1440 8 3 2880 1440 9 3 3360 1440 18 3 7680 1440
19 3 8160 1440 20 3 8640 1440 21 3 9120 1440 22 3 9600 1440 33 3 13440
1440 34 3 13920 1440 35 3 14400 1440 36 3 14880 1440 45 3 19200 1440 46
3 19680 1440 47 3 20160 1440 48 3 20640 1440 49 3 21120 1440 50 3 21600
1440 51 3 22080 1440 52 3 22560 1440 57 3 24960 1440 58 3 25440 1440 59
3 25920 1440 60 3 26400 1440 68 3 30720 1440 69 3 31200 1440 70 3 31680
```

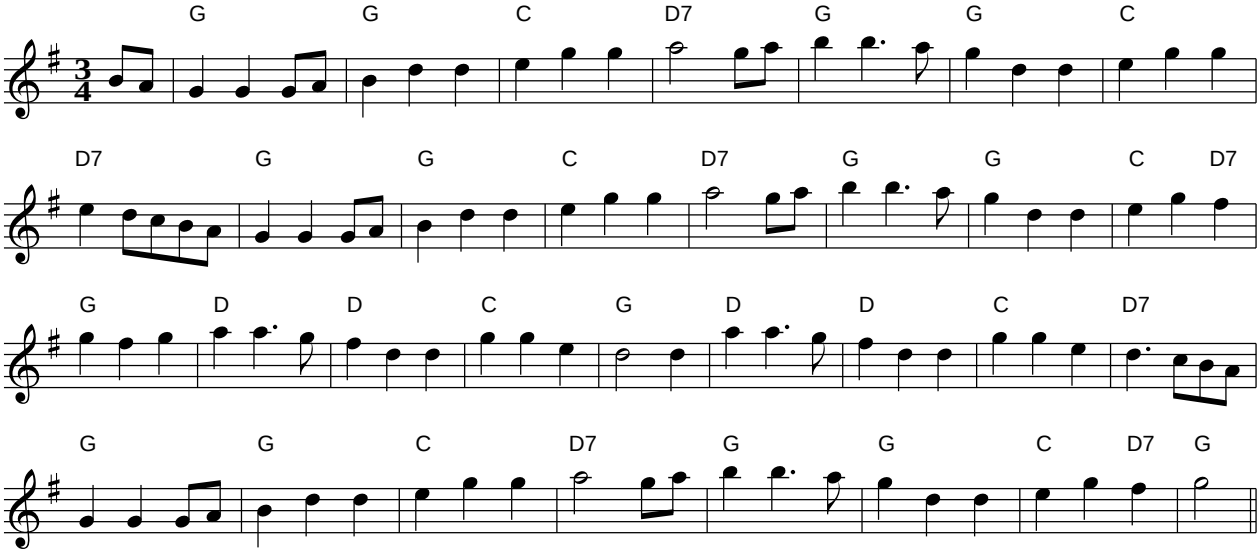
```
1440 71 3 32160 1440 82 3 36480 1440 83 3 36960 1440 84 3 37440 1440 85
3 37920 1440 94 3 42240 1440 95 3 42720 1440 96 3 43200 1440 97 3 43680
1440 0 0 0 1 1 1 0 0 -3 -3 -5 -8 -3 -3 -5 -8 0 -3 -3 -5 -8 -3 -3 -5 -8 -7
-8 -7 -7 -3 -3 -8 -7 -3 -3 -8 0 -3 -3 -5 -8 -3 -3 -5 480 480 480 100000
100000 0 0 0
29 6 4 1920 1920 7 4 2400 1920 8 4 2880 1920 18 4 7680 1920 19 4 8160
1920 20 4 8640 1920 21 4 9120 1920 33 4 13440 1920 34 4 13920 1920 35
4 14400 1920 45 4 19200 1920 46 4 19680 1920 47 4 20160 1920 48 4 20640
1920 49 4 21120 1920 50 4 21600 1920 51 4 22080 1920 57 4 24960 1920 58
4 25440 1920 59 4 25920 1920 68 4 30720 1920 69 4 31200 1920 70 4 31680
1920 82 4 36480 1920 83 4 36960 1920 84 4 37440 1920 94 4 42240 1920 95
4 42720 1920 96 4 43200 1920 0 0 0 1 1 1 0 0 -3 -3 -8 -3 -3 -5 0 -3 -3
-8 -3 -3 -5 -8 -7 -8 -7 -3 -3 -7 -3 -3 0 -3 -3 -8 -3 -3 480 480 480 480
100000 100000 100000 0 0 0 0
21 6 5 1920 2400 7 5 2400 2400 18 5 7680 2400 19 5 8160 2400 20 5 8640
2400 33 5 13440 2400 34 5 13920 2400 45 5 19200 2400 46 5 19680 2400 47
5 20160 2400 48 5 20640 2400 49 5 21120 2400 50 5 21600 2400 57 5 24960
2400 58 5 25440 2400 68 5 30720 2400 69 5 31200 2400 82 5 36480 2400 83
5 36960 2400 94 5 42240 2400 95 5 42720 2400 0 0 0 1 1 1 0 0 -3 -8 -3 -3
0 -3 -8 -3 -3 -5 -8 -7 -7 -3 -7 -3 0 -3 -8 -3 480 480 480 480 480 100000
100000 100000 100000 0 0 0 0 0
5 15 9 6240 4320 42 9 17760 4320 54 9 23520 4320 65 9 29280 4320 91 9
40800 4320 0 0 0 1 1 1 0 0 0 2 2 0 480 720 240 480 480 480 480 480 480
150000 33333 200000 100000 100000 100000 100000 100000 0 0 0 0 0 0 0 0 0
6 15 3 6240 1440 42 3 17760 1440 54 3 23520 1440 65 3 29280 1440 73 3
33120 1440 91 3 40800 1440 0 0 0 1 1 1 0 0 0 2 2 7 0 480 720 240 150000
33333 0 0 0
5 18 7 7680 3360 45 7 19200 3360 46 7 19680 3360 47 7 20160 3360 48 7
20640 3360 0 0 0 1 1 1 0 0 0 5 5 3 480 480 480 480 480 480 480 100000
100000 100000 100000 100000 100000 100000 0 0 0 0 0 0 0
3 45 8 19200 3840 46 8 19680 3840 47 8 20160 3840 0 0 0 1 1 1 0 0 5 5 480
480 480 480 480 480 480 480 100000 100000 100000 100000 100000 100000
100000 0 0 0 0 0 0 0 0
2 45 9 19200 4320 46 9 19680 4320 0 0 0 1 1 1 0 0 5 480 480 480 480 480
480 480 480 480 100000 100000 100000 100000 100000 100000 100000 100000 0
```

```
0 0 0 0 0 0 0 0
2 49 8 21120 3840 68 8 30720 3840 0 0 0 1 1 1 0 0 1 480 480 480 480 480
480 720 240 100000 100000 100000 100000 100000 150000 33333 0 0 0 0 0 0 0
0
3 53 4 23040 1920 64 4 28800 1920 72 4 32640 1920 0 0 0 1 1 1 0 0 5 0 480
480 720 240 100000 150000 33333 0 0 0 0
2 53 10 23040 4800 64 10 28800 4800 0 0 0 1 1 1 0 0 5 480 480 720 240
480 480 480 480 480 480 100000 150000 33333 200000 100000 100000 100000
100000 100000 0 0 0 0 0 0 0 0 0 0
2 54 10 23520 5280 91 10 40800 5280 0 0 0 1 1 1 0 0 -2 480 720 240 480
480 480 480 480 480 960 150000 33333 200000 100000 100000 100000 100000
100000 200000 0 0 0 0 0 0 0 0 0 0
```

B.4.3 Contraintes utilisateur

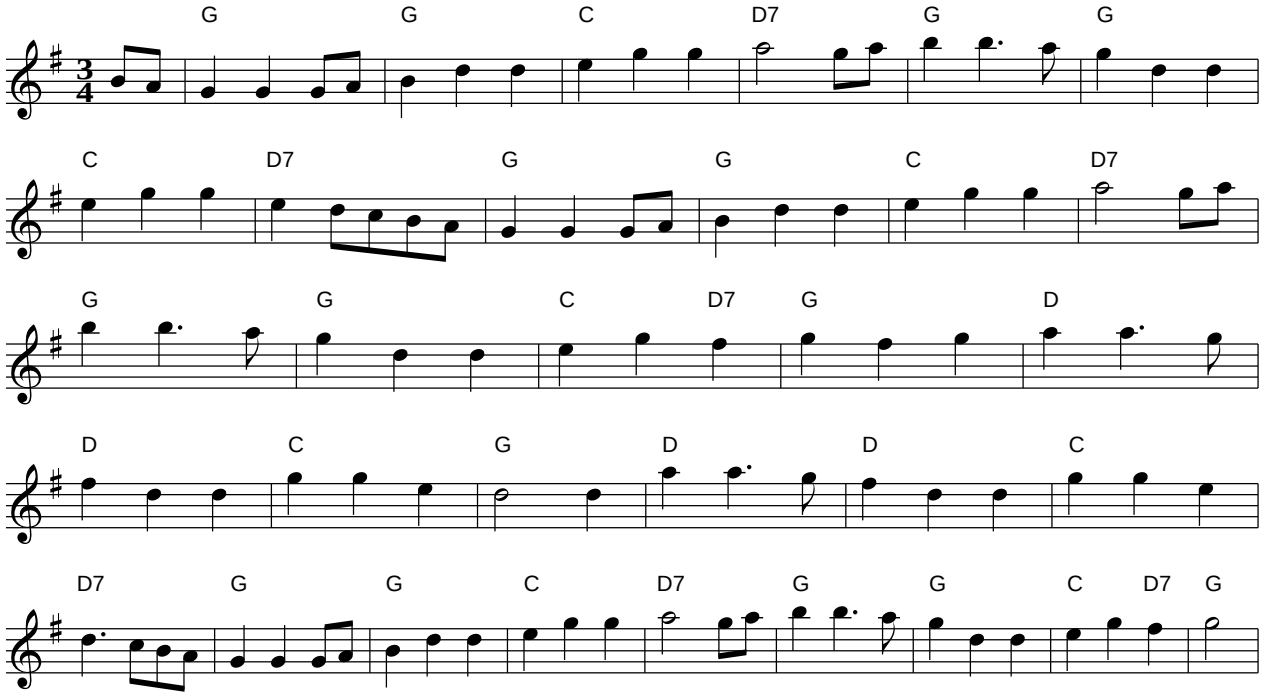
```
W S A W
phase1
phase2
```

B.4.4 Valse d'origine



```
X:12
T:Down in the Mines
S:Kevin Briggs
M:3/4
L:1/4
K:G
B/2A/2|"G"GGG/2A/2|"G"Bdd|"C"egg|"D7"a2g/2a/2|"G"bb3/2a/2|"G"gdd|"C"egg|
"D7"ed/2c/2B/2A/2|"G"GGG/2A/2|"G"Bdd|"C"egg|"D7"a2g/2a/2|"G"bb3/2a/2|"G"gdd|
"C"eg"D7"f|
"G"gfg|"D"aa3/2g/2|"D"fdd|"C"gge|"G"d2d|"D"aa3/2g/2|"D"fdd|"C"gge|
"D7"d3/2c/2B/2A/2|
"G"GGG/2A/2|"G"Bdd|"C"egg|"D7"a2g/2a/2|"G"bb3/2a/2|"G"gdd|"C"eg"D7"f|"G"g2||
```

B.4.5 Valse générée




```
X:1
T:Original waltz
R:Waltz
```

S:melody generation constraint system
M:3/4
L:1/4
K:G
B/2A/2|"G"GGG/2A/2|"G"Bdd|"C"egg|"D7"a2g/2a/2|"G"bb3/2a/2|"G"gdd|"C"egg|"D7"
ed/2c/2B/2A/2|"G"GGG/2A/2|"G"Bdd|"C"egg|"D7"a2g/2a/2|"G"bb3/2a/2|"G"gdd|"C"e
g"D7"f|"G"gfg|"D"aa3/2g/2|"D"fdd|"C"gge|"G"d2d|"D"aa3/2g/2|"D"fdd|"C"gge|"D7"d3/2c
/2B/2A/2|"G"GGG/2A/2|"G"Bdd|"C"egg|"D7"a2g/2a/2|"G"bb3/2a/2|"G"gdd|"C"eg"D7"
f|"G"g2|