Projet
Connaissance et Raisonnements

- Rapport de projet CR -
CentraleSupélec - Université Paris-Saclay, Gif-Sur-Yvette
23 février 2025

DAVID Erwan – FAYNOT Guillaume

Objectif du projet

La génération automatique de musique repose souvent sur des modèles probabilistes ou neuronaux. Dans ce projet, nous explorons une approche différente en utilisant plusieurs types de solveurs, notamment SAT, pour composer de la musique sous contraintes. L'objectif est de modéliser la génération musicale sous forme de problème de satisfiabilité logique, résolu ensuite par un solveur SAT. Cette approche permet de garantir des contraintes harmoniques et rythmiques tout en produisant des mélodies structurées. Nous explorerons par la suite d'autres types de solveurs pour améliorer l'harmonie musicale de la partition générée, notamment pseudo-booléenne et programmation linéaire.

Table des matières

I. Configuration et cadre musical

Le fichier config.py fixe un cadre musical qui sert de trame à la partition générée par le solveur. Nous fixons 7 notes sur une gamme en Do majeur, pour 30 temps joués. Deux pianos joueront deux partitions en même temps : le piano 1 joue l'octave 4 et le piano 2 joue l'octave 3. Nous avons donc au maximum 2 notes joués par instant. Ces contraintes ont été choisies pour leur capacité à produire des mélodies musicalement cohérentes tout en restant computationnellement gérables. L'équilibre entre contraintes harmoniques et rythmiques permet d'éviter à la fois la monotonie et le chaos musical.

Nous allons explorer plusieurs types de modélisation : SAT, pseudo-booléenne, et programmation linéaire mixte avec des entiers. Ces modélisations sont plus ou moins permissives et nous explorerons les possibilités offertes par chacune dans un contexte musical.

Notre travail peut être retrouvé sur Github. Le dépôt contient un README avec les explications permettent de lancer le logiciel.

II. Modélisation SAT

1. Pipeline génération de musique avec un solveur SAT

Le pipeline de génération musicale comprend plusieurs étapes :

1. Génération du fichier CNF : Un script Python synth.py génère un fichier CNF contenant un ensemble de clauses logiques représentant les contraintes musicales.
2. Résolution SAT : Le solveur SAT (Gophersat) est exécuté sur le fichier CNF pour trouver une assignation satisfaisante des variables. Les résultats sont stockés dans un fichier txt.
3. Génération du fichier MIDI : Le script gen.py traduit la solution SAT en notes MIDI exploitables.
4. Lecture du fichier MIDI : Le script play.py permet d'écouter la mélodie générée.

2. Génération SAT

La génération musicale repose sur une modélisation en logique propositionnelle, résolue par un solveur SAT garantissant le respect strict des règles musicales. Chaque note jouable est représentée par une variable booléenne, où une affectation vraie indique que la note est jouée à un instant donné. L'ensemble du problème est formulé sous forme de clauses en CNF, qui imposent des contraintes musicales garantissant la cohérence et l'harmonie de la mélodie générée.
Ces variables sont définies comme suit : $x_{t,n,i}$ avec :

- $x_{t,n,i} = 1$ si l'instrument i joue la note n à l'instant t
- $x_{t,n,i} = 0$ sinon.

Les contraintes sont exprimées sous forme de clauses logiques :

- Éviter les grands sauts : Une note ne peut pas changer brusquement de plus de 4 demi-tons entre deux instants successifs :

$$\forall t, \forall i, \forall n_1, \forall n_2, |n_2 - n_1| > 4 \Rightarrow \neg x_{t,n_1,i} \vee \neg x_{t+1,n_2,i}$$

- Harmonisation avec des accords (I, IV, V en Do majeur) : Certaines notes sont forcées à respecter une progression harmonique :

$$\forall t, \exists (n_1, n_2, n_3) \in \{I, IV, V\}, x_{t,n_1,i} \vee x_{t,n_2,i} \vee x_{t,n_3,i}$$

- Éviter la répétition excessive : Une note ne doit pas se répéter consécutivement pour un même instrument

$$\forall t, \forall i, \forall n, \neg x_{t,n,i} \vee \neg x_{t+1,n,i}$$

- Distribution des notes entre les instruments : Chaque instrument joue des notes dans une tessiture adaptée (octave différente pour le piano 1 et le piano 2), en s'assurant qu'à chaque instant , chaque instrument joue au plus une note :

$$\forall t, \forall i, \sum_n x_{t,n,i} \leq 1$$

- Et enfin, générer $x \in N$ intervalles aléatoires où les instruments jouent ensemble.

Ces contraintes sont transformées en clauses SAT par le synth.py et résolues par le solveur, permettant de générer une mélodie conforme aux règles de l'harmonie musicale.

III. Modélisation pseudo-booléenne

Ce modèle diffère du premier par sa résolution et son fonctionnement, cependant nous avons utilisé des contraintes similaires pour avoir des résultats similaires.
Voici les contraintes :

- Affectation Unique, on joue au maximum une note par temp :

$$\forall t, i, \sum_{n=0}^{N-1} x_{t,n,i} \leq 1$$

- Évitement des Grands Sauts : Pour éviter les sauts de plus de 4 intervalles entre deux temps consécutifs, on impose :

$$\forall t, \forall i, \forall n_1, \forall n_2, |n_2 - n_1| > 4 \Rightarrow x_{t,n_1,i} + x_{t+1,n_2,i} \leq 1$$

- Harmonisation par Accords : La progression d'accords est générée aléatoirement :

$$ChordProgression = \{C_t\}_{t=0}^{T-1} \; avec \; C_t \in \{I, IV, V, VI\}$$

  En plus, des intervalles aléatoires (définis via together_intervals) déterminent si les instruments jouent simultanément ou avec un décalage, en imposant pour t pair :

$$\sum_{n \in C_t} x_{t,n,0} \geq 1 \; et \; \sum_{n \in C_t} x_{t+\delta,n,1} \geq 1$$

  où $\delta = 0$ si t est dans un intervalle "ensemble" et $\delta = 1$ sinon.

- Évitement des Répétitions : Pour empêcher la répétition d'une note à distance $X = 2$:

$$\forall t, \forall i, \forall n, x_{t,n,i} + x_{t+2,n,i} \leq 1$$

- Contraintes Unitaires Aléatoires : Pour certains instants choisis aléatoirement, on force explicitement l'activation d'une note pour chaque instrument :

$$x_{t,n,i}^{rand} = 1$$

  Cela permet de diversifier davantage la mélodie.

Cette approche pseudo-booléenne offre plus de flexibilité dans l'expression des contraintes numériques par rapport au modèle SAT pur, tout en maintenant une complexité de résolution raisonnable.

Pour faciliter la résolution, nous utilisons Gurobi qui a l'avantage de posséder une syntaxe plus agréable que Gophersat. Le pipeline est le même que pour le solveur SAT, sauf qu'il exécute le fichier synthPB.py

IV. Programmation linéaire mixte

Finalement, voici la résolution MILP. Les contraintes utilisées sont :

- Jouer exactement une note par temps. L'objectif est d'avoir un air différent :

$$\forall t, i, \sum_{n=0}^{N-1} x_{t,n,i} = 1$$

- Évitement des Grands Sauts : Pour éviter les sauts de plus de 4 intervalles entre deux temps consécutifs, on impose :

$$\forall t, \forall i, \forall n_1, \forall n_2, |n_2 - n_1| > 4 \Rightarrow x_{t,n_1,i} + x_{t+1,n_2,i} \leq 1 + slack_{t,n_1,n_2,i}$$

  Avec le $slack$ qui est une variable de relaxation introduite pour autoriser exceptionnellement les grands sauts, mais inflige une forte pénalité dans l'objectif

- Harmonisation par Accords : La progression d'accords est générée aléatoirement :

$$ChordProgression = \{C_t\}_{t=0}^{T-1} \; avec \; C_t \in \{I, IV, V, VI\}$$

- Évitement des Répétitions : Pour empêcher la répétition d'une note à distance $X = 2$:

$$\forall t, \forall i, \forall n, x_{t,n,i} + x_{t+2,n,i} \leq 1$$

- Transitions harmonieuses entre les notes
  - Modélisation de la hauteur : Des variables $y_{t,i}$ (de type entier) sont introduites pour représenter la valeur MIDI de la note jouée par l'instrument i à l'instant t. La contrainte lie $y_{t,i}$ aux variables x via la NOTE_MAPPING :

$$y_{t,i} = \sum_{n=0}^{N-1} NOTEMAPPING_{i,n} . x_{t,n,i}$$

  - Modélisation de la transition : des variables $d_{t,i}$ (continues, $d \geq 0$) représentent la différence absolue $|y_{t+1,i} - y_{t,i}|$.Cette relation est encadrée par deux contraintes linéaires :

$$y_{t+1,i} - y_{t,i} \leq d_{t,i}$$
$$y_{t,i} - y_{t+1,i} \leq d_{t,i}$$

  - Objectif partiel : Pour encourager des transitions douces entre les notes, il faut minimiser :

$$\sum_{t,i} d_{t,i}$$

- Objectif global : minimiser la somme des écarts de transitions $d_{t,i}$ (pour favoriser la douceur des transitions) et d'y ajouter une pénalité très élevée (ici, 1000 fois la somme des $slack$) afin d'éviter les grands sauts non désirés.

$$\sum_{t,i} d_{t,i} + 1000 \sum_{t,n_1,n_2,i} slack_{t,n_1,n_2,i}$$

Le choix du MILP permet une optimisation plus fine des transitions mélodiques grâce à l'introduction de variables continues, au prix d'une complexité de calcul plus élevée. Les pénalités importantes sur les grands sauts assurent que la solution préfère naturellement les transitions douces.

Le pipeline est le même, en utilisant cette fois ci le script synthMILP.py.

V. Musiques uniques

Chaque modèle intègre des mécanismes d'aléatoirisation pour garantir la génération de mélodies différentes à chaque exécution :

1. SAT solveur

- Introduction de contraintes unitaires aléatoires via random_cnf() qui force certaines notes à être jouées à des moments spécifiques, évitant ainsi la répétition des mêmes solutions
- Génération aléatoire de la progression d'accords parmi les accords majeurs (I, IV, V)
- Création d'intervalles aléatoires où les deux pianos jouent simultanément, permettant des variations dans l'interaction entre les instruments

2. Modèle Pseudo-Booléen

- Utilisation d'une progression d'accords générée aléatoirement sur l'ensemble de la partition
- Définition dynamique des moments de jeu simultané via together_intervals, créant des variations dans la structure temporelle de la pièce

3. Modèle MILP

- Intégration d'une progression harmonique aléatoire utilisant quatre accords (I, IV, V, vi), enrichissant les possibilités harmoniques
- La progression aléatoire influence directement l'optimisation des transitions, produisant des chemins mélodiques différents à chaque exécution

Cette approche par randomisation, combinée aux contraintes structurelles, permet d'obtenir des mélodies qui respectent les règles musicales tout en évitant la génération répétitive de solutions identiques. Le degré de liberté laissé au solveur entre ces contraintes aléatoires assure un bon équilibre entre variété et cohérence musicale.

VI. Pour aller plus loin – IA hybride

Enfin, un court essai a été produit, basé sur le TD sur l'IA hybride a été réalisé. Il permet de mettre en perspective un raisonnement afin d'établir des préférences de musique. En effet, on pourrait imaginer un processus au sein d'un groupe pour choisir un genre musical qui satisfait au maximum les préférences des membres, et exploiter différentes contraintes qui caractérisent ce genre musical pour générer une musique que l'on peut classer dans le genre choisi.

VII. Conclusion

Ce projet démontre la viabilité d'une approche par contraintes pour la génération musicale automatique. Les trois méthodes implémentées (SAT, pseudo-booléenne et MILP) offrent différents compromis entre expressivité des contraintes et efficacité de résolution. Les résultats obtenus montrent que la formalisation logique des règles musicales permet de générer des mélodies structurées et harmonieuses. Les travaux futurs pourraient explorer l'extension du modèle à des structures musicales plus complexes et l'intégration de contraintes stylistiques additionnelles.