Prototype de génération procédurale de contrepoints à la manière de Jean-Sébastien Bach

Jérémie Roux, Violaine Prince

PROTOTYPE DE GÉNÉRATION PROCÉDURALE DE CONTREPOINTS À LA MANIÈRE DE JEAN-SÉBASTIEN BACH

Jérémie Roux
Université de Montpellier (étudiant)
jeremie.roux@etu.umontpellier.fr

Violaine Prince
LIRMM - Équipe TEXTE
prince@lirmm.fr

RÉSUMÉ

L'objectif de cet article est de présenter un ensemble d'algorithmes permettant la génération automatique de fugues à la manière de Jean-Sébastien Bach. Il s'intéressera principalement aux différents constituants de la fugue : sujet, réponse, contre-sujet et épisode, pour lesquels des algorithmes de génération, définis selon les règles de *l'Art de la Fugue* [2], seront décrits et commentés. La séquence des constituants, fondée sur une analyse des fugues du *Clavier Bien Tempéré I* par l'équipe *Algomus*, est représentée sous la forme d'un graphe de transition (à la manière d'un modèle Markov caché) également détaillé. On s'intéressera également à la comparaison entre un modèle de transition des éléments des fugues selon la tradition de l'analyse musicale, et un autre issu de l'analyse statistique du corpus d'*Algomus*.

1. INTRODUCTION

La musique est un art que l'on peut qualifier de scientifique par sa construction, du fait de l'organisation décidée par le compositeur de l'enchaînement des rythmes et des notes, correspondant ou pas au style d'écriture de son époque. Certains compositeurs comme *Johann Sebastian Bach* (1685-1750) ont poussé cette conception de la musique à un point que l'on considère toujours aujourd'hui comme sortant de l'ordinaire. En particulier, lorsque l'on analyse *L'Art de la Fugue* [2], on ne peut s'empêcher de penser qu'il existe un certain nombre d'algorithmes implicites qui sont employés par le compositeur pour réaliser ses compositions (certains y voyaient une allusion aux mathématiques les plus "strictes" [4]). Cette régularité remarquable a conduit les informaticiens que nous sommes à nous poser deux questions.

Premièrement, est-il possible de transformer les algorithmes musicaux de Bach en algorithmes "informatiques", c'est-à-dire de les formaliser et de les transposer dans des ordinateurs, alors que la théorie musicale sous-jacente est éminemment complexe ? Et si oui, quel serait le prix de cette transformation, dans le sens de la simplification, de l'abandon éventuel de certains aspects, étant données les contraintes informatiques de la gestion de la complexité ?

Deuxièmement, si une telle action était possible, elle le serait sous forme d'un premier prototype destiné à évoluer incrémentalement avec chaque "victoire" sur la complexité. Mais elle aurait l'avantage de proposer des choix algorithmiques dans la continuité d'une mélodie donnée. Les oeuvres de *L'Art de la Fugue* sont des configurations particulières qui ont fixé les choix effectués par Bach au moment où il les a écrites. Ne pourrait-on pas simuler automatiquement, grâce à ce prototype, d'autres configurations qui réaliseraient, pour chaque étape, d'autres choix, et du coup, fournir de la matière première intéressante pour une analyse plus poussée des oeuvres en question ? Ce serait alors un usage musicologique inédit, un peu à la manière de ce qu'apportent des simulations informatiques à d'autres disciplines, en explorant l'arbre des possibles.

Dans le travail entrepris ici, nous nous sommes donnés pour objectif de réaliser une étude de type "preuve de concept", sachant que de précédentes avancées sur la modélisation pour la composition ont été largement passées au crible dans des travaux comme [11], s'appuyant entre autres sur le traité fondateur de Barbaud [12]. Pour cela, nous avons développé des algorithmes afin de théoriser et produire des morceaux sur la base d'une mélodie donnée en utilisant spécifiquement l'écriture contrapuntique de Bach dans [2]. Ces fameux algorithmes musicaux sont une *expression de l'écriture contrapuntique,* et celle de Bach est riche tout en obéissant à des règles strictes. Le contrepoint selon Bach est qualifié de *luxuriant* par Bougeret [3], qui l'oppose à l'écriture contrapuntique relatée dans le traité de Fux ([6]). Ce dernier s'appuie sur les pratiques non tonales de la fin de la Renaissance (Palestrina, Gesualdo) à laquelle furent formés les compositeurs de l'époque classique (tout en s'y opposant).

Le **contrepoint** est défini comme une forme d'écriture musicale qui trouve ses origines avec la polyphonie et qui consiste en la superposition organisée de lignes mélodiques distinctes (une définition fort didactique en est donnée dans [3]). Nous étudions les dimensions mélodiques, rythmiques et harmoniques de cette écriture, dont la **fugue** est le meilleur représentant.

Nous proposons ici un ensemble d'algorithmes de génération de constituants d'une fugue, selon notre inter-

prétation computationnelle de l'écriture de Bach, ainsi qu'une structure générale d'enchaînement des constituants, fondée sur un modèle de Markov caché [1] . Une ébauche musicologique de ce modèle est fournie puis complétée à partir d'une analyse automatique des fugues du *Clavier Bien Tempéré I* proposée par l'équipe *Algomus* des universités de Lille et d'Amiens [8], [7].

2. UN ÉTAT DE L'ART TRÈS SITUÉ

La musicologie a très longuement analysé les fugues et ce, tout au long des siècles. Le *Traité de la Fugue* de Gédalge [1] en donne une bonne définition, en particulier de la *fugue d'école*, qui est le principe à partir duquel on enseigne cet art aux étudiants en écriture. Les composants de la fugue auxquels nous nous référons dans les sections suivantes s'appuient sur des éléments issus de la pratique musicologique. Les travaux de Czaczkes [10] se focalisent davantage sur les fugues du *Clavier Bien Tempéré*, qui est un répertoire riche et sur lequel nous nous appuierons puisqu'il est au coeur de notre objectif (*i.e*, l'écriture contrapuntique de Bach), et aussi parce qu'une analyse computationnelle en a été faite et dont nous parlerons longuement.

En ce qui concerne l'informatique musicale et la génération de fugues, les travaux de Jan Philipp Hakenberg et Mathias Müller, auteurs de *The Pirate Fugue* avaient un objectif très proche du nôtre, et rien moins qu'audacieux : écrire le troisième livre (et non écrit par Bach) du *Clavier Bien Tempéré* ! La méthodologie en était cependant très différente : "*Rather than relying on encoding music theory into an algorithm, our approach is data driven : using a collection of over 5000+ digital classical scores...*" [2] . Les auteurs du logiciel se réfèrent également aux travaux de David Cope qui utilise des ATN (*augmented transition networks*) et une approche de type "informatique linguistique" pour générer automatiquement des compositions "dans le style" d'un compositeur ou un autre (dont des chorals de Bach) [5].

Les travaux qui nous ont personnellement le plus directement inspirés et ceux sur lesquels nous nous sommes appuyés sont les suivants : en analyse, les travaux de Giraud *et al.* se sont centrés depuis 2012 sur l'analyse computationnelle des fugues de Bach [8] en vue de la détection automatisée de motifs récurrents tels que *sujets* et *contre-sujets* (définis dans la section suivante) ainsi qu'*épisodes* pour en arriver à la définition d'un environnement complet d'analyse partagée [7] permettant à tout chercheur intéressé de pouvoir utiliser ces ressources en apprentissage comme en modélisation. En génération, outre ceux de Cope, les travaux les plus proches de notre problématique sont ceux de Hadjeres *et al.* en 2017 [9], qui ont proposé un système d'apprentissage automatique, fondé sur des réseaux de neurones, nommé *DeepBach*, entraîné par un apprentissage préalable sur des cantates de J. S. Bach. Il a été développé pour harmoniser automatiquement des thèmes musicaux dans le style du "choral", sachant que certains chorals comprennent des éléments canoniques, qui sont une sorte d'état initial des fugues.

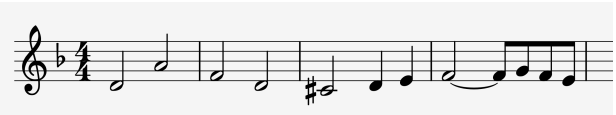
Notre travail se trouve donc à la croisée des deux démarches : sur le même matériau que les travaux d'*Algomus*, nous avons construit un premier prototype de génération de fugue, ainsi complémentaire de l'analyse réalisée et fondé sur ses résultats pour alimenter notre structure d'enchaînement. Avec une démarche ayant le même objectif que *DeepBach*, mais aussi que [5] et *The Pirat Fugue*, nous nous sommes intéressés à la génération de fugues en optant pour un graphe basé sur la structure d'un modèle de Markov pour représenter les transitions entre différents éléments constituants des fugues, transitions dont les valeurs (ou poids) sont issues des travaux d'analyse des fugues [8] et en écrivant localement des algorithmes de génération pour chaque constituant. En ce sens, les travaux de Cope furent quelque peu précurseurs puisque l'ATN est aujourd'hui remplacé par les modèles de Markov dans les disciplines qui les utilisent.

3. ALGORITHMES ET ÉCRITURE CONTRAPUNTIQUE

Nous définissons et modélisons ici les différents éléments de l'écriture des fugues. Pour des raisons de simplification des algorithmes, nous nous en tiendrons à des fugues à 2 voix.

3.1. Sujet d'une fugue

Le **sujet** d'une fugue est le thème, motif mélodique et rythmique principal du morceau qui sera répété à l'identique, avec des variations et/ou des transpositions tout au long de l'œuvre. On donne donc en entrée du programme le sujet de la fugue que l'on veut obtenir ainsi que la tonalité de ce sujet (**Figure 1**).



Légende : b = blanche, n = noire, c = croche, $+$ = dièse, i = bémol

```
1 D m
2 D4_b A4_b F4_b D4_b C+4_b D4_n E4_n F4_2.5 G4_c F4_c E4_c
```

Figure 1: Sujet du premier contrepoint de L'Art De La Fugue (*BWV 1080*) en *Ré mineur* de J. S. Bach et fichier associé donné en entrée du programme

Le programme ne garde pas ce fichier tel quel, il crée une **voix** (tableau composé d'objets de type *note*) et chaque note de ce tableau correspond aux notes du sujet données en entrée du programme et conservées dans le même

1 . Lien vers le dépôt Git (code source et exemples) : *https://gitlab.etu.umontpellier.fr/jeremieroux/bach*

2 . *The Pirat Fugue : http://djtascha.de/the-pirate-fugues/*

ordre. Dans le codage du sujet présenté en **Figure 1**, les notes sont séparées par des espaces et l'information de la *hauteur* de la note est séparée par un '_' de l'information sur sa *longueur* (l'information de la longueur de la note dans un fichier donné en entrée du programme peut être codée par la première lettre du rythme correspondant mais aussi par un nombre de temps). Pour des raisons de conception, on ne peut donner que des sujets de fugue d'une longueur divisible par 4, c'est-à-dire qui remplissent complètement une suite de mesures de signature rythmique 4/4. On garde également en mémoire la tonalité du sujet qui est la tonalité principale du morceau.

3.2. La réponse, répétition transposée du sujet

La fugue démarre par une exposition du sujet en solo où l'on place le sujet donné en entrée sans le modifier sur une des voix disponibles. Il s'agit d'une voix plutôt grave si on veut qu'il soit répété à une hauteur plus aiguë, et inversement.

Tout de suite après l'exposition du sujet, il est répété à l'identique mais avec éventuellement une transposition (le plus souvent à la quarte ou la quinte, supérieure ou inférieure) sur une voix différente de celle sur laquelle il a été exposé : c'est la **réponse** [3] . Si le sujet est de longueur l, la réponse est donc de longueur l et la totalité de la longueur du morceau à l'instant où la réponse est terminée est de longueur $2l$ (**Figure 2**).

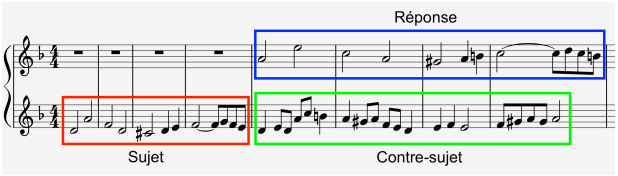


Figure 2: Réécriture du premier contrepoint de L'Art De La Fugue (*BWV 1080*) en *Ré mineur* de J. S. Bach avec le sujet original (la réponse et le contre-sujet sont générés par notre programme)

Algorithme 1 : EcrireReponse(S : sujet, t : nombre de demi-tons) : sujet transposé

pour chaque *note n* **de** S **faire**
  $n.hauteur \leftarrow n.hauteur + t$;
fin
renvoyer S;

Il n'y a donc pas de chevauchement entre le sujet et la réponse, la réponse commence sur le premier temps de la mesure d'après (et sur une autre voix) où le sujet termine. On implémente l'écriture de la réponse sur la voix 2 (la voix 1 étant par convention celle où le sujet est exposé en premier) comme décrit par l'**Algorithme 1**. On a bien la réponse qui commence après l'exposition du sujet, transposée de t demi-tons (variable qui peut être négative ou positive selon si la transposition est respectivement plus grave ou plus aiguë que l'original).

3 . Pour simplifier notre prototype, on effectue une réponse avec transposition exacte du sujet et non une réponse tonale avec mutation mélodique de la réponse (qui sert à conserver la même tonalité malgré le changement de degré de la réponse par rapport au sujet).

3.3. Le contre-sujet, accompagnement de la réponse

Le **contre-sujet** (**Figure 2**) est complémentaire mélodiquement et rythmiquement à la réponse. Il est sur la même voix que le sujet, de la même longueur et dans son prolongement, *i.e*, le passage du sujet au contre-sujet doit être imperceptible.

3.3.1. Complémentarité rythmique avec la réponse

L'essence du contrepoint est le flot rythmique ininterrompu et régulier, c'est-à-dire que les motifs rythmiques des voix sont complémentaires et les écouter en même temps donne l'impression que la musique ne s'arrête pas, certains rythmes ont lieu au même moment pour créer un accord et amplifier la polyphonie.

C'est dans ce but que l'on va définir une liste des motifs rythmiques pour la réponse et proposer des rythmes complémentaires pour l'accompagner. Il y aura parfois plusieurs motifs rythmiques complémentaires possibles pour le même motif initial : il faudra alors choisir de façon aléatoire, ce qui fait que deux contrepoints sur le même sujet de fugue pourront être différents. Pour simplifier le code, un motif rythmique est une succession de rythmes au sein d'une même mesure à 4 temps (c'est pourquoi la somme des rythmes pour chaque motif fait toujours 4).

```
2.0,2.0 | 1.0,0.5,0.5,0.5,0.5,1.0 - 1.0,1.0,1.0,1.0
2.0,1.0,1.0 | 1.0,1.0,2.0
1.0,1.0,1.0,1.0 | 2.0,2.0
4.0 | 1.0,1.0,1.0,1.0 - 2.0,2.0
2.0,0.5,1.5 | 1.5,0.5,2.0
2.5,0.5,0.5,0.5 | 0.5,0.5,0.5,0.5,2.0
```

Figure 3: Motifs rythmiques sur une mesure et leur(s) complémentaire(s)

La **Figure 3** indique quelques motifs rythmiques (à gauche de '|') et leurs complémentaires (à droite de '|', séparés par des tirets) que l'utilisateur peut modifier (motifs stockés dans un fichier texte). Pour chaque mesure, c'est à dire chaque motif rythmique de 4 temps, on prend au hasard une de ses mesures complémentaires que l'on place à la suite d'une liste de rythmes, le tout formant alors les rythmes de notre contre-sujet (voir **Algorithme 2**).

3.3.2. Enchaînement mélodique des notes dans le contre-sujet et harmonie avec la réponse

On définit l'**intervalle harmonique** comme étant la distance entre deux notes jouées simultanément et un **intervalle mélodique** comme étant la distance entre deux notes jouées l'une après l'autre. Un intervalle harmonique

Algorithme 2 : EcrireRythmeCS(S : sujet, $contreMesure$: tableau de durées de notes en **Figure 3**) : rythmes du contre-sujet

Variables : r : rythmes du contre-sujet
$r \leftarrow []$;
pour chaque *mesure* m *de* S **faire**
  $i \leftarrow aleatoire(0, taille(contreMesure[m]) - 1)$;
  $r \leftarrow ajouterFin(contreMesure[m][i])$;
fin
renvoyer r;

est toujours positif ($i_h = |h_1 - h_2|$) mais un intervalle mélodique est positif si la première note est plus grave que la deuxième et négatif si la première note est plus aiguë que la seconde ($i_m = h_2 - h_1$). *Les intervalles sont mesurés en demi-tons.*

```
1 vector<int> gammeMajeure = {0,2,4,5,7,9,11};
2 vector<int> gammeMineure = {0,2,3,5,7,8,11};
3 vector<int> lIntervalH = {3,4,5,7};
4 vector<int> lIntervalM = {-2,2,1,-1,-3,5,3,4,-4,-5,0};
5 vector<int> lIntervalM2 = {-3,-2,4,-4,1,-1,5,3,-5,2,0}; (Marche Harmo)
```

Figure 4: Extrait du code C++ présentant les intervalles harmoniques et mélodiques prioritaires et les intervalles de chaque degré des gammes (majeure ou mineure) avec la tonique (intervalles en demi-tons)

Dans la **Figure 4**, les variables *gammeMajeure* et *gammeMineure* représentent les distances mélodiques de chaque note des gammes à leur tonique respective. On obtient les gammes dans les tonalités voulues en ajoutant la valeur de la hauteur de la tonique en demi-ton par rapport à *Do* (par exemple, $\forall n \in$ *gammeMajeure*, $\forall x \in \mathbb{N}^*$, $n + 5 + 12x$ sont toutes les notes de la gamme de *Fa Majeur* sur toutes les octaves car 5 correspond à *Fa*).

La variable *lIntervalH* correspond à la liste de tous les intervalles harmoniques possibles pour deux notes données (tierce mineure et majeure, quarte et quinte qui sont plutôt consonants). Les listes *lIntervalM* et *lIntervalM2* sont deux ordres de préférence d'intervalles mélodiques pour deux notes données. On privilégie les petits intervalles mélodiques (on ne discute pas ici du signe) car ils sont plus naturels dans une composition, une mélodie étant généralement composée de notes qui se suivent en hauteur, les intervalles plus grands étant utilisés si les petits ne sont pas possibles (approche intuitive). Pour un rythme fixé on obtient donc toujours la même génération de contre-sujet.

L'**Algorithme 3** permet de savoir quelle est la note sur une voix à un instant t (t-ième temps du morceau). Cette fonction sera très utile pour avoir les informations sur une note dont on veut calculer un intervalle harmonique avec son homologue, *i.e*, la note simultanée sur l'autre voix.

Ensuite, l'**Algorithme 4** va choisir des hauteurs pour les notes du contre-sujet (sur les rythmes préalablement calculés) en respectant les contraintes sur la tonalité du morceau, les intervalles harmoniques cohérents avec les notes de la réponse et l'ordre de priorité des intervalles mélodiques (les plus petits d'abord, les plus grands ensuite, l'unisson en dernier) pour garantir une uniformité de la mélodie du contre-sujet et de son harmonie avec la réponse.

Algorithme 3 : NoteSelonTemps($voix$: tableau de notes, t : durée) : note

Variables : i : entier, $sommeT$: durée
$i \leftarrow 0$; `// Variable qui contiendra le numéro de la note sur la voix`
$sommeT \leftarrow 1.0$; `// Variable qui contient la somme des rythmes des notes visitées, début sur le` **`premier temps`** `de la mesure 1`
tant que *sommeT* $<$ *t* **faire**
  $sommeT \leftarrow sommeT + voix[i].rythme$;
  $i \leftarrow i + 1$;
fin
si *sommeT* $\neq$ *t* **alors**
  $i \leftarrow i - 1$;
fin
renvoyer $voix[i]$;

Algorithme 4 : EcrireHauteurCS(CS : sortie de l'**Algorithme 2**, R : sortie de l'**Algorithme 1**) : contre-sujet

Variables : CS : contre-sujet
pour chaque *note* n **de** $rythmeCS$ **faire**
  $m \leftarrow$ note correspondante dans R;
  $nPred \leftarrow$ note précédente dans CS ou dans S si c'est la première note;
  pour chaque im **de** $lIntervalM$ **faire**
    si *compatible(nPred+im,m,*$lIntervalH$*)* **alors**
      si $im \neq 0$ **alors**
        $n.hauteur \leftarrow nPred.hauteur + im$
      sinon
        rallonger $nPred$;
      fin
    sinon
      $n \leftarrow$ silence;
    fin
  fin
fin
renvoyer CS;

La variable *lGammeMode* correspond à la liste des intervalles à la tonique dans la gamme et le mode choisi. Les données *lIntervalM* et *lIntervalH* sont respectivement l'ordre de préférence des intervalles mélodiques et harmoniques. Pour chaque intervalle mélodique, on vérifie si la note est bien dans la gamme correspondant à la tonalité du morceau et si elle est bien en harmonie avec son homologue sur l'autre voix. On choisit la première qui est sous ces contraintes. Si la hauteur choisie est la même que

la précédente (unisson mélodique), on fusionne ces deux notes pour en donner une seule dont la longueur est la somme. Si on ne trouve pas de hauteur possible sous ces contraintes, on écrit un silence de longueur égale à la note que l'on aurait dû placer.

3.4. Marche harmonique

Une **marche harmonique** est un motif rythmique et mélodique (parfois commun à plusieurs voix) appelé **modèle**, répété plusieurs fois de suite mais transposé à chaque répétition (ou reproduction) qui peut servir de transition entre des répétitions de sujets et de contre-sujets. Elle peut appartenir à ce que l'on appelle le **divertissement** (voir **Figure 5**). Lorsque le motif est répété mais transposé sur un degré plus aigu, on parle de *marche harmonique ascendante*. Lorsqu'au contraire il est de plus en plus grave, c'est une *marche harmonique descendante* (**Figure 6**). Chaque reproduction du modèle est généralement répétée un ton ou un demi-ton plus haut ou plus bas que la précédente selon le sens de la marche choisi [4]. Le nombre de reproductions sera un entier aléatoire entre 2 et 4 car c'est ce nombre là qu'on trouve généralement dans les fugues de J. S. Bach, ce qui équivaut à des marches harmoniques de longueur 3 à 5 car on a $l_{marche} = n_{reproduction} + 1$).

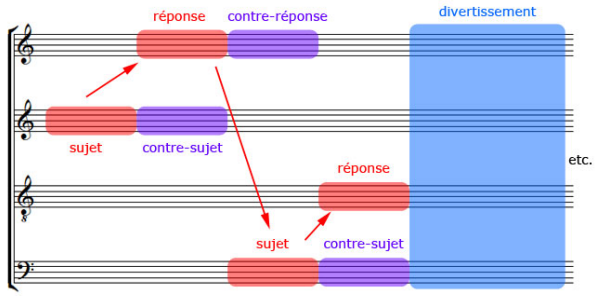


Figure 5: Schéma simplifié d'une fugue

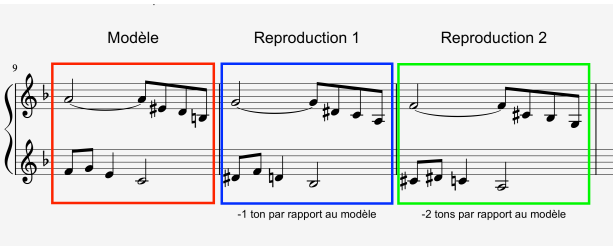


Figure 6: Exemple d'une marche harmonique descendante (par tons) générée automatiquement avec un motif complémentaire sur deux voix (sur la base du premier rythme de la **Figure 7**)

Pour simplifier dans le cas d'une fugue à 2 voix, on considère que le motif fait 4 temps (une mesure pleine).

4 . Dans le cadre de notre prototype, on génère une marche harmonique non-modulante. Si on avait voulu améliorer notre algorithme pour générer des marches modulantes, on aurait fait en sorte de moduler dans la bonne tonalité à chaque reproduction.

```
0.5,0.5,1.,2. | 2.5,0.5,0.5,0.5
0.5,0.5,1.5,0.5,1. | 1.5,0.5,1.5,0.5
0.5,0.5,0.5,0.5,2. | 1.,1.5,0.5,0.5,0.5
```

Figure 7: Motifs rythmiques de marche harmonique sur une mesure et leur complémentaire

Algorithme 5 : EcrireMarcheH($voix1$, $voix2$: tableaux de notes, $lGammeMode$, $lIntervalM2$, $lIntervalH$: liste de hauteurs, $motifMarcheH$: map<listes de durées de notes, listes de listes de durées de notes>) : tableau de tableaux de notes

Variables : $lHautGamme$: liste de hauteurs de notes,
$decalage$,r,v,nb : entiers, $interv$,h,i : hauteur,
$rythmeV1$,$rythmeV2$: liste de durées de notes,
$nPred$,n : note, $motif1$, $motif2$: tableau de notes

$lHautGamme \leftarrow lGammeMode$;
pour chaque *entier* e **de** $lHautGamme$ **faire**
  $lHautGamme \leftarrow lGammeMode + hauteurTonique$;
  // Attribution de la bonne gamme
fin
$decalage \leftarrow aleatoireEntre(0,1)$; // Sens de la marche harmonique
$interv \leftarrow aleatoireEntre(1,2)$; // Intervalle de reproduction de la marche harmonique
si $decalage = 1$ **alors**
  $decalage \leftarrow interv$; // Marche harmonique montante
sinon
  $decalage \leftarrow -interv$; // Marche harmonique descendante
fin
$r \leftarrow aleatoireEntre(0, motifMarcheH.taille - 1)$;
// Choix aléatoire du motif de la marche
$v \leftarrow aleatoireEntre(0,1)$; // Choix aléatoire de la voix pour le motif ou son complémentaire
$rythmeV1 \leftarrow motifMarcheH[r][v]$;
$rythmeV2 \leftarrow motifMarcheH[r][1-v]$;
$nb \leftarrow aleatoireEntre(2,4)$; // Choix aléatoire du nombre de reproductions
pour chaque *flottant* f **de** $rythmeV2$ **faire**
  $nPred \leftarrow voix2[voix2.taille - 1]$; // Note précédant la marche sur la voix 2
  $n \leftarrow nouvelleNote()$;
  $n.rythme \leftarrow f$;
  $h \leftarrow nPred.hauteur$;
  tant que $(h = nPred.hauteur)$ ***ou***
  $(h \bmod 12 \notin lHautGamme)$ **faire**
    $i \leftarrow aleatoireEntre(0, lIntervalM.taille - 1)$;
    // Intervalle mélodique aléatoire
    $h \leftarrow nPred.hauteur + lIntervalM$; // Calcul de la nouvelle hauteur
  fin
  $n.hauteur \leftarrow h$; // Attribution de la bonne hauteur à la note
  $ajouterFin(voix2, n)$;
  $ajouterFin(motif2, n)$; // Extraction du motif sur la voix 2
  $nPred \leftarrow n$;
fin
$voix1 \leftarrow EcrireHauteurCS(voix1, voix2, lGammeMode, lIntervalM, lIntervalH, 4, tonique)$;
pour i ***de*** $rythmeV1.taille - 1$ ***à*** 0 **faire**
  $ajouterFin(motif1, voix1[voix1.taille - 1 - i])$;
  // Extraction du motif sur la voix 1
fin
renvoyer
$EcrireReprod(voix1, voix2, motif1, motif2, decalage, nb)$; // Reproductions

On construit une liste de motifs sur deux voix (**Figure 7**) séparés par un 'l' pour ce qui est des motifs complémentaires d'une même marche et séparés par un retour à ligne pour les différents éléments de cette liste. On choisit aléatoirement le sens de la marche (montante ou des-

cendante), l'intervalle de reproduction (un demi-ton ou un ton), le nombre de reproductions (de 2 à 4), le motif de la marche et l'attribution d'une partie de ce motif à une voix et l'autre partie à la seconde voix. Ensuite, on attribue une mélodie sur ces rythmes pour la voix qui vient d'énoncer la réponse. On choisit cette mélodie en sélectionnant un intervalle mélodique aléatoire qui n'est pas un unisson pour chaque note que l'on place. On écrit ensuite l'autre voix de la même manière qu'on a attribué des hauteurs de note lors de la génération du contre-sujet (en prenant comme réponse la partie de la marche harmonique déjà attribuée). On a alors écrit le modèle de notre marche harmonique (déroulé sur **Algorithme 5**).

3.5. Suite du contrepoint, succession de sujets, contre-sujets, réponses et marches harmoniques

Une **fugue**, vue sous l'angle structurel ici privilégié, est donc une succession d'entrées de sujets, de réponses et de contre-sujets sur un certain nombre de voix. On considère qu'un sujet ne peut être ré-exposé à l'identique, transposé et/ou avec de légères variations de notes et de rythmes que lorsque la réponse de l'exposition précédente suivie d'une marche harmonique est terminée. Sur 2 voix, ce principe parait trivial mais il l'est beaucoup moins lorsqu'il s'agit de faire la même chose pour un plus grand nombre de voix. La **Figure 8** représente les premières séquences de la toccata et fugue BWV 538 [5] .

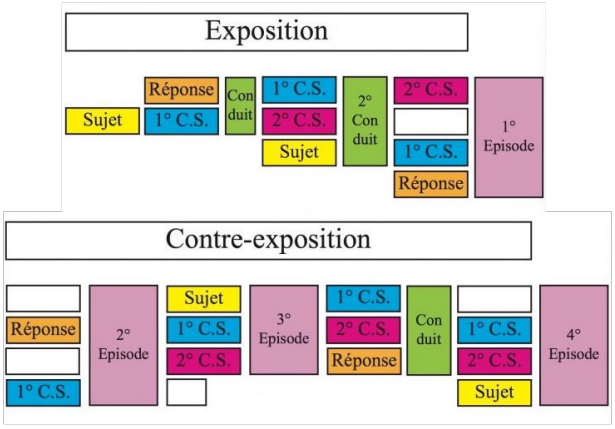


Figure 8: Plan du début de la fugue à 4 voix issue de la Toccata et Fugue en *Ré mineur* (*BWV 538*) de J. S. Bach, extrait de *https://jplecaudey.com/analyse-musicale/bwv-538-toccata-e-fuga-in-d/*

Un **conduit** correspond à une marche harmonique et fait le lien entre deux expositions du sujet voire parfois entre un sujet et contre-sujet ou encore entre deux contre-sujets. Les **épisodes** sont des segments de la composition qui peuvent eux aussi contenir des marches harmoniques mais également des méthodes d'écriture non abordées ici mais qui ne sont pas moins programmables (comme par exemple des passages à la tonalité relative).

Les *motifs* (rythme et hauteur de notes) composant le modèle sur les 2 voix sont stockés dans les variables *motif1* et *motif2*. L'**Algorithme 6** détaille comment le motif est répété *nb* fois (nombre de reproductions) sur les 2 voix en prenant en compte le décalage et en transposant de plus en plus le motif original. Les notes résultant de ces reproductions successives sont ajoutées aux tableaux de notes correspondant aux mélodies des 2 voix. L'algorithme renvoie un tableau contenant les notes des deux voix.

Algorithme 6 : EcrireReproductions($voix1$, $voix2$,$motif1$,$motif2$: tableaux de notes, $decalage$: hauteur, nb : entier) : tableau de tableaux de notes

Variables : $decalageTotal$: hauteur, n : note
$decalageTotal \leftarrow 0$; `// Initialisation du décalage`
pour *i* ***de*** 1 ***à*** nb **faire**
  $decalageTotal \leftarrow decalageTotal + decalage$;
  `// Incrémentation du décalage`
  pour chaque *note* $n1$ **de** $motif1$ **faire**
    $n \leftarrow nouvelleNote()$;
    $n.rythme \leftarrow n1.rythme$;
    `// Conservation du même rythme sur la voix 1`
    $n.hauteur \leftarrow n1.hauteur + decalageTotal$;
    `// Transposition du motif 1`
    $ajouterFin(voix1, n)$;
  fin
  pour chaque *note* $n2$ **de** $motif2$ **faire**
    $n \leftarrow nouvelleNote()$;
    $n.rythme \leftarrow n2.rythme$;
    `// Conservation du même rythme sur la voix 2`
    $n.hauteur \leftarrow n2.hauteur + decalageTotal$;
    `// Transposition du motif 2`
    $ajouterFin(voix2, n)$;
  fin
fin
renvoyer $nouveauTableau(voix1; voix2)$;

Dans les fugues plus simples (**Figure 5**), on a bien sûr l'exposition du sujet, la réponse et le contre-sujet (parfois une contre-réponse qui est complémentaire au sujet non transposé). La suite de l'oeuvre est appelée *divertissement* (terme déjà abordé lors de la définition des marches harmoniques) et on n'y retrouve pas de motifs clairement liés au sujet, il s'agit plutôt là de variations comme des marches harmoniques isolées ou d'autres techniques d'écriture que nous ne traiterons pas ici.

3.6. Fin d'un contrepoint

Les longueurs des fugues tout comme le nombre de voix sont variables, allant de quelques dizaines de mesures à parfois des centaines. On donne au programme la contrainte de terminer le morceau après un certain nombre de mesures (le programme peut dépasser ce nombre de

5 . Pour écouter l'œuvre : *https://youtu.be/ZRY7zrMGCi8?t=311*

mesures le temps de terminer le motif qu'il est en train de générer). On veut conclure le morceau sur une **cadence parfaite**, c'est à dire un enchaînement du cinquième degré au premier degré. Lorsque la tonalité d'un morceau est dans le mode mineur, il est de coutume (dans la musique baroque notamment) de terminer le morceau sur la même tonique mais dans le mode majeur. C'est ce que l'on appelle une **tierce picarde**.

3.7. Discussion sur les algorithmes

On remarque que dans la **Figure 9**, la complexité de l'écriture de la fugue complète est de $O(t^{nbVoix})$ (avec t le nombre maximum de notes par voix et $nbVoix$ le nombre de voix). Lorsqu'on travaille avec deux voix (comme tout au long de cette étude), la complexité est de $O(t^2)$. La complexité de notre algorithme d'écriture de contrepoints est donc raisonnable pour un faible nombre de voix mais explose si on en ajoute d'autres. De plus, on ne peut pas facilement adapter notre algorithme pour plus de 2 voix car il ne suffit pas d'appliquer les contraintes sur les voix 2 à 2, il faut le faire sur toutes les voix en même temps (notamment les contraintes liées aux intervalles harmoniques).

Plusieurs de ces complexités dépendent de la longueur des listes comprenant les intervalles mélodiques et harmoniques ainsi que le nombre de notes dans la gamme. Ces termes étant constants, on pourrait les simplifier dans l'expression des complexités pour clarifier l'écriture (c'est le cas des **Algorithmes 4, 5 et 5'**).

	Nom	Structure	Composante note	Complexité
1	EcrireReponse	Réponse	Rythme/Hauteur	$O(n)$
2	EcrireRythmeContreSujet	Contre-sujet	Rythme	$O(m)$
3	NoteSelonTemps		$\emptyset$	$O(t)$
4	EcrireHauteurContreSujet		Hauteur	$O(m * i_{\text{mel}} * i_{\text{har}} * t_{\text{gam}})$
5	EcrireMarcheHarmonique	Marche harmonique	Rythme/Hauteur	$O(f * i_{\text{mel}} * t_{\text{gam}})$
5'	EcrireModele			$O(f * i_{\text{mel}} * t_{\text{gam}})$
6	EcrireReproductions			$O(\text{nb} * f)$
7	EcrireFugue	Fugue entière		$O(t^{\text{nbVoix}})$

Légende

- n est le nombre de notes dans le sujet
- m est le nombre de notes dans le contre-sujet
- t est le nombre de notes sur la voix jusque-là (au maximum, nombre de notes sur toute la voix)
- i_{mel} est le nombre d'intervalles mélodiques
- i_{har} est le nombre d'intervalles harmoniques
- t_{gamme} est le nombre de notes dans la gamme de la tonalité choisie
- f est le nombre maximal de notes sur un motif de marche harmonique
- nb est le nombre de reproductions d'une marche harmonique (au maximum 4)
- $nbVoix$ est le nombre de voix

Figure 9: Complexité des différents algorithmes étudiés (l'algorithme 5' est l'algorithme 5 sans compter l'appel à l'algorithme 6)

4. THÉORISATION GÉNÉRALE D'UN CONTREPOINT PAR UN HMM

Pour théoriser plus facilement et fidèlement les différents éléments d'une fugue, on construit une ébauche de modèle de Markov caché ou HMM (pour *Hidden Markov Model*) (**Figure 10**) élaboré à partir d'une analyse musicale manuelle traditionnelle de quelques fugues du *Clavier Bien Tempéré I*. L'**Algorithme 7** utilise cet HMM et choisit de prendre telle ou telle transition vers un autre état (chaque état est un algorithme comme ceux qui ont été détaillés précédemment). L'état final est atteint lorsque le nombre de mesures escompté pour le morceau est égalé ou légèrement dépassé.

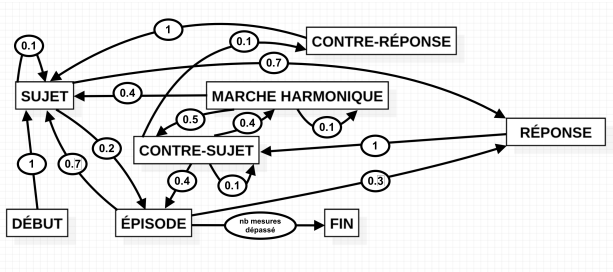


Figure 10: Ébauche de modèle de Markov caché d'une fugue

Algorithme 7 : EcrireFugue($voix1$,$voix2$, $sujet$: tableaux de notes, $nbMesuresMax$: entier, $modele$: HMM) : tableau de tableaux de notes

Variables : $nbMesures$: entier, n : note, $etatHMM$: état en cours dans le HMM, $terminaison$: booléen

$nbMesures \leftarrow 0$; `// Initialisation du nombre de mesures déjà écrites`

$etatHMM \leftarrow DEBUT$; `// Initialisation de l'état HMM`

$terminaison \leftarrow faux$;

tant que *non(terminaison)* **faire**

  $executerEtat(etatHMM, modele)$;

  $nbMesures \leftarrow nbMesures + nbMesuresCreees()$; `// Incrémentation du nombre de mesures`

  $etatHMM \leftarrow choisirEtatSuivant(etatHMM, modele)$;

  si $(etatHMM = EPISODE)$ ***et*** $nbMesures \geq nbMesuresMax$ **alors**

    $etatHMM \leftarrow FIN$;

    $executerEtat(etatHMM)$; `// Composer la fin du morceau`

    $terminaison \leftarrow vrai$;

  fin

fin

renvoyer $nouveauTableau(voix1; voix2)$;

Afin d'affiner les valeurs des transitions du modèle, nous analyserons dans la prochaine section les données publiées par Giraud *et al.* dans les articles déjà cités, et tenterons d'amender le modèle traditionnel avec les résultats de l'analyse statistique du corpus de fugues.

5. ANALYSE ET UTILISATION DES DONNÉES RÉCOLTÉES PAR L'ÉQUIPE ALGOMUS

La plateforme d'annotation et de visualisation *Dezrann* [6] proposé par Giraud *et al.* permet les actions suivantes :

- la reconnaissance des différents éléments d'une fugue
- une étude de la superposition des éléments (entre les voix)
- une étude de la succession des éléments
- une étude fréquentielle des transitions possibles entre éléments.

Nous avons réalisé l'ensemble de ces actions sur les 24 fugues fournies par l'équipe, mais pour des raisons de concentration sur le modèle HMM proposé préalablement, nous nous limiterons à l'étude des transitions pour lesquelles nous avions défini des valeurs à partir d'observations de faible ampleur.

Les données à analyser se présentent sous forme d'un fichier JSON pour chacune des 24 fugues : il comprend la liste des éléments identifiés dans la fugue concernée avec pour chacun d'entre-eux le type, le temps de départ (peut commencer sur un quart de temps ou un demi-temps), leur durée, le numéro de la voix sur laquelle il est (compté de haut en bas) et des informations plus précises sur l'élément en question.

5.1. Analyse de la transition entre les éléments d'une fugue

On cherche à étudier la manière dont les éléments précédemment évoqués (*i.e*, les sujets, réponses *etc.*) se succèdent dans les fugues du *Clavier Bien Tempéré*. Pour cela on écrit différents algorithmes qui ne sont pas détaillés ici [7].

On récolte tout d'abord les statistiques sur les transitions entre les éléments constituants des fugues 2 à 2 ainsi que lorsque l'un de ces éléments est le dernier de la voix ($'FIN'$). On note le nombre total de transitions depuis chaque élément dans la valeur $'TOTAL'$. Cet algorithme récolte donc les données pour les transitions au départ d'un élément e pour toutes les voix de l'ensemble des 24 fugues. Un autre algorithme permet de créer une matrice d'entiers en appliquant le précédent algorithme autant de fois que d'éléments à prendre en considération. Ces entiers représentent le nombre de transitions entre un élément et un autre. On utilise les données précédentes pour calculer les probabilités de transition (tableau de flottants). Pour chaque case, il fait le calcul (1) avec x et y 2 éléments, $x \rightarrow y$ la transition de x vers y et $x \rightarrow$ les transitions depuis x. On obtient ainsi une matrice de transition entre les différents éléments constituant une fugue que l'on va par la suite représenter par un HMM.

$$P(x \rightarrow y) = \frac{nb(x \rightarrow y)}{nb(x \rightarrow)} \quad (1)$$

5.2. Discussion sur ces résultats et comparaison avec le modèle traditionnel

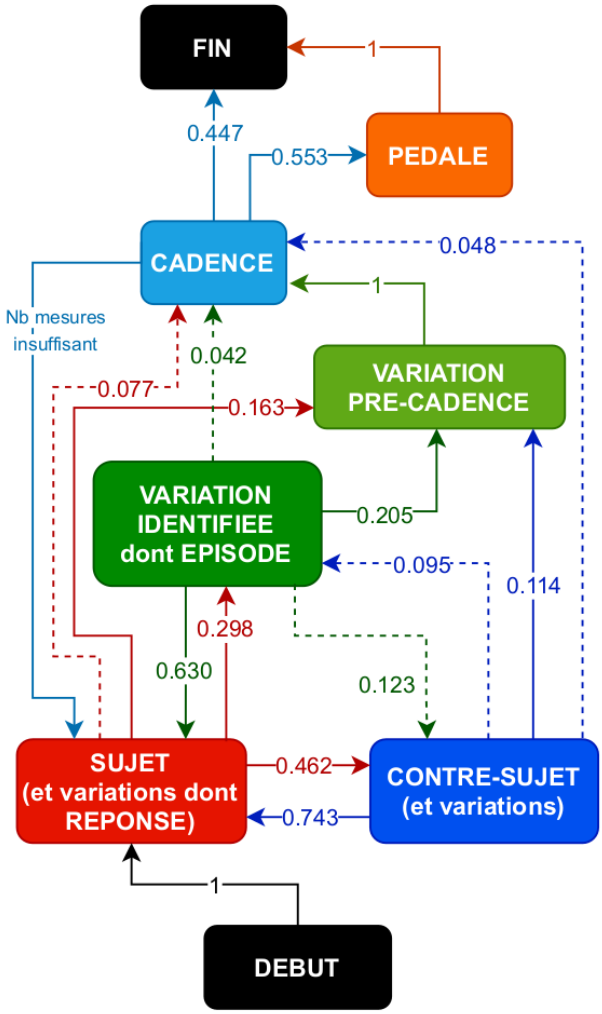


Figure 11: Modèle de Markov caché (HMM) des transitions entre les différents éléments d'une fugue selon l'analyse des fugues du *Clavier Bien Tempéré I* par Giraud *et al.*

L'environnement *Dezrann* modifie complètement le modèle traditionnel général de structuration et d'enchaînement de la fugue pour la génération. Tous les éléments modificatifs ne sont pas ici reportés, mais il est intéressant de voir ce qui, à configuration similaire, a été véritablement mis à jour. Pour cela, nous avons refait une représentation en graphe de notre HMM en calculant les valeurs des transitions à partir des données de Giraud *et al.* (voir **Figure 11**). Nous nous sommes autorisés à retourner depuis une cadence vers l'exposition d'un sujet (ou variation du sujet) à condition que le nombre de mesures minimal (donné par l'utilisateur) n'ait pas été atteint. On

6 . http://algomus.fr/dezrann/, http://www.dezrann.net/

7 . Le détail de ces algorithmes est disponible sur le *Git* : *https://gitlab.etu.umontpellier.fr/jeremieroux/bach* notamment sur la version 4 du rapport

a également modifié le HMM de façon qu'il n'y ait pas de boucle (transition d'un état x vers lui même) : ce problème réside sans doute dans le fait que l'algorithme de reconnaissance indexe les variations des éléments et que lorsque l'on supprime ces indexations (pour la clarté du HMM) on a deux éléments identiques qui se succèdent sur la même voix. Nous avons aussi confondu toutes les variations d'un même élément pour des raisons d'identification mais aussi pour pouvoir jouer plus tard sur l'évolution de l'élément original au cours de la musique en définissant d'autres ordres et règles propres à chaque élément. Les probabilités ne sont donc plus vraiment comparables même si l'on remarque en avoir sous-estimé quelques-unes comme le retour (sur une voix donnée) d'un contre-sujet vers un sujet (et leurs variations) que l'on peut expliquer par la présence dans le corpus de retours au sujet sans développement préalable. Cette analyse a aussi permis d'identifier d'autres éléments comme les pédales et les cadences ainsi que leurs transitions associées.

5.3. Génération de squelettes de fugues selon le HMM de transition des éléments calculé

On utilise maintenant les données présentées en **Figure 11** pour générer des squelettes de fugue sous le même format que les fichiers fournis par Giraud *et al.* L'**Algorithme 8** étage l'entrée des sujets et fait se succéder les divers éléments. On crée une matrice de probabilités (**Figure 12**) qui intègre les données du HMM dans le code. La **Figure 13** est un exemple de structure de fugue ainsi générée qui étage l'entrée des sujets et fait se succéder les éléments.

```
vector<string> elem = {"S", "CS", "Cad", "Ped", "Var", "?", "FIN"};
vector<vector<float> > trans =
  {{0.000, 0.462, 0.077, 0.000, 0.298, 0.163, 0.000},
   {0.743, 0.000, 0.048, 0.000, 0.095, 0.114, 0.000},
   {0.000, 0.000, 0.000, 0.553, 0.000, 0.000, 0.447},
   {0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 1.000},
   {0.630, 0.123, 0.041, 0.000, 0.000, 0.205, 0.000},
   {0.000, 0.000, 1.000, 0.000, 0.000, 0.000, 0.000}};
```

Figure 12: Tableau des probabilités de transitions et du nom des éléments correspondants

```
vide vide vide   S  CS   S   ?  Var   S  Var  Cad  FIN
vide   S  Var   S  CS   S  Var   S   ?  Var  Cad  FIN
  S    ?  Cad  Var   ?  Var   ?  Var   S  Var  Cad  FIN
vide vide   S   CS   S  CS   S   CS   S  Var  Cad  FIN
```

Figure 13: Exemple de génération d'un squelette de fugue à 4 voix

Algorithme 8 : GénérationSquelette($ordre$: ordre d'entrée des voix, $elem$: liste de string, $trans$: tableau 2d de flottants) : tableau 2d de string

Variables : F : tableau 2d de flottants, lg : entier, $nbElemMin$: entier, $suivant$: string
$F \leftarrow creerTableau2d$(taille($ordre$),0); // Création d'un tableau du bon nombre de lignes vides
$lg \leftarrow 0$; // Nombre d'éléments sur chaque voix
$nbElemMin \leftarrow 7$; // Nombre minimal d'éléments sur chaque voix (alternative au nombre de mesures)
/* Etagement de l'entrée des sujets */
tant que *lg<taille(ordre)* **faire**
  pour i ***de*** 1 ***à*** *taille(ordre)-1* **faire**
    si i=*lg* **alors**
      ajouter(F[$ordre[i]$],'S');
    sinon
      si i<*lg* **alors**
        ajouter(F[$ordre[i]$],'vide');
      sinon
        ajouter(F[$ordre[i]$],elemTrans(F[$ordre[i]$][-1],elem,trans));
      fin
    fin
  fin
  $lg \leftarrow lg + 1$;
fin
/* Suite après l'étagement */
tant que *F[0][-1]* $\neq$ *'FIN'* **faire**
  pour i ***de*** 1 ***à*** *taille(ordre)-1* **faire**
    /* On calcule l'élément suivant */
    $suivant \leftarrow$elemTrans(F[$ordre[i]$][-1],elem,trans);
    si $suivant$='*Ped*' || $suivant$='*FIN*' **alors**
      si lg<$nbElemMin$ **alors**
        ajouter(F[$ordre[i]$],'S');
      sinon
        /* Fin de la fugue */
        ajouterToutesLignes(F,$suivant$);
        si $suivant$='*Ped*' **alors**
          ajouterToutesLignes(F,'FIN');
        fin
        $i \leftarrow$taille($ordre$)-1;
      fin
    sinon
      ajouter(F[$ordre[i]$],$suivant$)
    fin
  fin
  $lg \leftarrow lg + 1$;
fin
renvoyer F;

6. CONCLUSION

La fugue est le type de morceau qui représente le mieux l'écriture du contrepoint chez Jean-Sébastien Bach. Polyphonique par essence, elle présente une structure qui se prête bien à une démarche algorithmique de construction, aussi bien qu'à une analyse automatique des motifs qui la constituent.

Notre travail, ici présent, a consisté à proposer un ensemble d'algorithmes d'écriture des éléments d'une fugue, à assembler pour réaliser un générateur automatique de fugues à la *J. S. Bach*. Après avoir brièvement passé en revue les travaux les plus proches de notre objectif, nous nous sommes centrés sur les éléments constitutifs des fugues de Bach. La section 3 a décrit ces différents constituants : le sujet, ensemble mélodique de base servant de motif initial, ainsi que les réponses (motifs complémentaires issus d'une action de transposition harmonique ou rythmique).

Nous avons montré comment on pouvait les représenter en machine, les coder avec un ensemble de contraintes qui ont été énoncées, et les installer sur des fugues à deux voix (limite due à des raisons de faisabilité pour l'état actuellement prototypal du générateur). Sujet et réponse sont aussi suivis de contre-sujets, contre-réponses, également générés. Nous avons aussi montré comment il était possible de produire ces "entre deux" que sont les divertissements, ou épisodes, au sein desquels nous avons cependant distingué un type particulier, la marche harmonique, que nous avons modélisée et ajoutée à notre bibliothèque d'algorithmes. L'enchaînement des différents constituants obéit aussi à un certain nombre de règles à modéliser en génération. La section 4 propose un modèle de Markov caché sous forme d'un graphe dont les noeuds sont les constituants et les arcs représentent des transitions ainsi que leurs probabilité de franchissement, réalisé à partir d'une analyse musicale manuelle traditionnelle. Afin d'affiner ce modèle, nous avons exploité les données de Giraud *et al.* de l'équipe *Algomus*, qui a produit un environnement fort intéressant pour l'analyse et la recherche de données des 24 fugues du livre I du *Clavier Bien Tempéré* de J.S Bach.

Ce qui était important pour nous était de voir comment cet environnement pouvait amender notre modèle d'enchaînement pour la génération. Plusieurs éléments peuvent être exploités, mais nous nous sommes ici surtout focalisés sur leur apport pour une meilleure définition des probabilités de transition. La section 5 montre justement comment ces données sont utilisées ainsi que la modification des valeurs des probabilités de transition entre les constituants. Nous nous sommes également aperçus que l'environnement ainsi exploité nous permettait réellement d'affiner la structure même du graphe d'enchaînement, et pas seulement les probabilités de transition. C'est pourquoi, l'étape prochaine du travail consistera à mettre à jour réellement l'ensemble du graphe d'enchaînement afin de le rendre plus riche et plus conforme aux réelles complexités des fugues de Bach. Ce travail ne peut être possible qu'à l'aide de l'environnement *Dezrann*, [7], car un passage par une analyse manuelle rendrait la tâche beaucoup plus coûteuse en efforts de modélisation et de transposition.

Toujours est-il que ce projet semble converger vers les bases d'un *générateur de fugues*, peut-être restreint dans un premier temps, mais néanmoins faisable. Ce dernier est fondé sur l'exploitation d'une bibliothèque algorithmique de génération de constituants de la fugue par un modèle de type probabiliste pour l'enchaînement desdits constituants. Rien n'empêchera par la suite, une fois le prototype construit, de prévoir des séquences possibles d'apprentissage et d'entraînement sur les fugues analysées par l'équipe *Algomus*, afin de conforter ses choix et/ou de les adapter pour une plus grande conformité avec *L'Art de la Fugue*.

7. REFERENCES

[1] Gédalge André. *Traité de la Fugue*. Enoch Cie, Paris, 1901.

[2] Jean-Sébastien Bach. L'Art de la Fugue (Die Kunst Der Fuge), 1740-1750. Bibliothèque d'État de Berlin.

[3] Gérard Bougeret. Harmonie, contrepoint, représentation : prélude à une didactique de l'écriture musicale. *Spirale. Revue de recherches en éducation, hors-série n4. Les représentations en formation(2), pp. 51-68*, 2005.

[4] Suzanne Clercx. *Le Baroque et la Musique : Essai d'Esthétique Musicale*. Bruxelles, 1948.

[5] David Cope. *Computers and Musical Style*. A-R Editions Inc, Maddison Wisconsin, 1991.

[6] Johann Joseph Fux. *Gradus ad Parnassus*. Johann Peter Van Ghelen, Wien, 1725.

[7] Mathieu Giraud, Richard Groult, and Emmanuel Leguy. Dezrann, a Web Framework to Share Music Analysis. *TENOR, pp. 104-110*, 2018.

[8] Mathieu Giraud, Richard Groult, Emmanuel Leguy, and Florence Levé. Computational Fugue Analysis. *Computer Music Journal, 39(2)*, 2015.

[9] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach : a Steerable Model for Bach Chorales Generation. *Proceedings of the 34th International Conference on Machine Learning, PMLR 70 :1362-1371*, 2017.

[10] Czaczkes Ludwig. *Analyse des Wohltemperierten Klaviers : Form und Aufbau der Fuge bei Bach*. Österreichischer Bundesverlag, Wien, München, 1965.

[11] Marcel Mesnage and André Riotte. Modélisation informatique de partitions, analyse et composition assistée. In *Cahiers de l'Ircam n 3, Recherche musicale : La composition assistée par ordinateur*, pages 1–1. Ircam - Centre Georges Pompidou, 1993. cote interne IRCAM : Mesnage93a.

[12] Barbaud Pierre. *Initiation à la composition musicale automatique*. Dunod Paris, 1965.