Contraintes et objets, clefs de voûte d'un outil d'aide à la composition ?

Philippe Ballesta

Projet SECOIA, INRIA-CERMICS Sophia-Antipolis,
2004, route des Lucioles, B.P. 93,
06902 SOPHIA-ANTIPOLIS Cedex

adresse électronique : ballesta@sophia.inria.fr

Résumé : *Poser un problème de composition musicale de manière déclarative impose la définition des structures musicales, puis celle des règles d'organisation que ces structures doivent vérifier. Un outil d'aide à la composition doit donc, non seulement permettre ces déclarations d'une manière naturelle et non réductrice, mais aussi proposer au compositeur la résolution efficace du problème qu'il s'est fixé. Pour répondre à cette double préoccupation, nous proposons de baser la réalisation d'un tel outil sur un formalisme mêlant idéalement programmation par contraintes et programmation par objets. Nous évaluerons la justesse de cette idée par le développement d'une application concrète d'aide à la réalisation d'exercices d'harmonie. Enfin, nous verrons que l'emploi de techniques d'optimisation utilisables en programmation par contraintes, concourt à l'amélioration de l'intérêt musical des solutions.*

Mots clés : précomposition, représentation des connaissances, outil d'aide à la composition, types, objets, contraintes, techniques de consistance, problème de satisfaction de contraintes (CSP), algorithme de *branch and bound*.

1 Aider le compositeur

Le but premier de la composition musicale est la production d'une œuvre musicale artistiquement achevée. Pour mener à bien une tâche si complexe, l'esprit humain a développé un certain nombre de méthodes ou techniques d'écriture [1]. L'Occident a vécu pendant plusieurs siècles dans le cadre de la pratique commune du système tonal, dont les règles fortes étaient appliquées par tous. Au vingtième siècle, chaque compositeur est plutôt désireux d'édicter son propre système, d'organiser le matériau musical selon ses propres principes ([Machover 85], [Courtot 92b]). Dans tous les cas, une part importante du travail de composition consiste donc en l'organisation de structures musicales selon un modèle, modèle tantôt partagé et éventuellement personnalisé par chaque compositeur, tantôt unique et complètement original.

1.1 Anatomie de la composition

L'ensemble d'idées et d'actions menant de l'intuition créatrice originelle à l'œuvre achevée, peut être schématiquement divisé en deux groupes.

1. La *précomposition* prend en charge l'application de toute technique compositionnelle visant à une organisation et une structuration du matériau musical, conformes toutes deux aux souhaits du compositeur ([Hoffstader 82], [Loy 88], [Philippot 85]). Cette recherche méthodique d'une formalisation est déclarative car elle passe préalablement par une définition des objets musicaux et des règles que ceux-ci doivent vérifier.

---

[1] Nous nous plaçons dans le cadre de la tradition musicale occidentale, dans lequel l'élaboration des œuvres est indissociable de leur traduction écrite. Voir [Billard 85] et [Molino 88].

Une solution à un tel problème est alors une combinaison d'objets musicaux vérifiant ces règles.

2. La *composition* apporte à l'œuvre sa dimension artistique et créatrice. Elle asservit le précédent groupe en se situant à la fois en amont et en aval de celui-ci. En amont, la définition ou le choix même des structures musicales et règles formelles à leur appliquer , constituent *ipso facto* des actes de création. En aval, le compositeur évalue selon des critères personnels, l'intérêt de chacun des résultats proposés par les méthodes précompositionnelles. Si la sélection des *meilleurs* candidats est formalisable, le compositeur peut déléguer la responsabilité de celle-ci à une nouvelle méthode qui sera incluse dans un groupe précompositionnel devenant ainsi de plus en plus sélectif. Néanmoins, ce dernier groupe n'a pour but généralement qu'un prétraitement formel que le talent artistique du compositeur devra transcender .

1.2 Un outil d'aide à la composition (OAC)

Comme l'ordinateur excelle à tout travail de structuration et d'organisation, on peut raisonnablement penser lui confier l'automatisation de l'étape de précomposition. Le but recherché est de délester le compositeur de tout un travail fastidieux en systématisant au maximum les tâches répétitives qui entrent dans la composition. Mener à bien une telle entreprise impose d'une part à l'artiste un effort conséquent et délicat de formalisation et de rationalisation de ses méthodes de création ([Arveiller 85], [Charbonnier 85]). D'autre part, il faudra surmonter un obstacle technologique en rendant la machine suffisamment *intelligente* pour qu'elle puisse s'adapter aux besoins du compositeur , et non l'inverse ([Machover 81]). Ainsi, cette médiation entre l'artiste et l'ordinateur comme moyen de représentation, ne peut être réussie que par l'emploi d'outils informatiques possédant certaines qualités.

La première qualité de l'outil devra être sa faculté d'adaptation au vocabulaire et aux idées du compositeur . Il faudra lui permettre de définir le plus naturellement possible les structures ou objets musicaux qu'il souhaite manipuler ainsi que les opérations, règles ou contraintes à leur appliquer .

La seconde qualité est conséquence de la vue déclarative portée sur les problèmes de composition musicale. Produire une ou l'ensemble des combinaisons d'objets musicaux vérifiant les règles d'organisation que le compositeur se fixe est une tâche coûteuse, si bien que la recherche de l'efficacité de l'outil informatique sera primordiale.

2 Quelles techniques informatiques utiliser ?

Quelles techniques informatiques employer pour bâtir un OAC possédant les deux précédentes qualités ?

2.1 Décrire sans réduction : une représentation par *types*

La vue déclarative d'un problème de composition musicale passe non seulement par la définition d'objets ou de structures que le compositeur désire manipuler , mais aussi par la définition des règles d'organisation que ceux-ci doivent satisfaire. Quel moyen de représentation choisir pour maximiser , au travers de ces définitions, la première qualité d'un OAC ?

F. Courtot ([Courtot 90], [Courtot 92a], [Courtot 92a]) choisit la notion de *type abstrait* comme base de représentation des connaissances ([Blevis 92]). Chaque objet musical est un type soit primitif, soit complexe s'il est résultat de l'application d'un opérateur de *composition* sur des types primitifs ou complexes. Ces opérateurs peuvent être des *constructeurs de listes* (superposition ou juxtaposition temporelle d'objets musicaux de type donné) ou des constructeurs de *taxonomies* (composition de plusieurs types, avec ou sans contraintes, pour en former de plus complexes). La notion de type abstrait sous-entend la présence de *proprié-*

tés ou *attributs* décrivant le type (notamment le domaine) et la définition d'*opérations* sur ces types (relation d'ordre, successeur, différence…). En outre, des opérateurs d'*association* clarifient le réseau de types (opérateur d'équivalence entre types, opérateur liant un type à un autre type représentant l'intervalle entre deux éléments du premier…).

Sur ce modèle très complet de représentation des objets musicaux par types, se greffe la définition des *actions* à effectuer sur ces objets. Courtot choisit de représenter celles-ci comme des *relations de contraintes* plutôt que des *fonctions*, arguant que le travail du compositeur consiste plus à mettre en relation des éléments qu'à en produire toujours dans le même sens. Cette opinion cadre tout à fait avec notre vision déclarative de la composition.

La saisie de telles relations conduit à l'obtention d'un réseau de dépendance entre types. A ce réseau correspond un programme *Prolog* dont l'exécution construit une ou l'ensemble des combinaisons d'objets vérifiant les relations de contraintes. L'emploi de mécanismes de retardement (la primitive *geler* de Prolog) ne permet pas d'éviter un usage passif des contraintes. Une contrainte codée sous la forme d'un prédicat gelé n'est en effet activée qu'après l'instanciation des objets sur lesquels elle porte.

Il est intéressant de noter que Courtot fournit tous les arguments qui justifient l'emploi de la programmation par contraintes : aspect relationnel plutôt que fonctionnel des liens entre objets musicaux, contraintes de composition ou d'association entre types, vision déclarative des problèmes de précomposition, recherche de l'efficacité.

2.2 Rechercher l'efficacité : une approche par *contraintes*

Dans ce type d'approche, on assimile l'étape de précomposition à la résolution d'un problème de *satisfaction de contraintes* (CSP). Un tel problème est défini par la donnée d'un ensemble de variables ayant chacune un domaine fini de valeurs, et par la donnée d'un ensemble de contraintes portant sur ces variables. Une solution est alors une combinaison des valeurs des variables qui satisfait toutes les contraintes. La représentation d'un problème de précomposition sous la forme d'un CSP est somme toute assez naturelle puisqu'elle découle de la vision déclarative portée sur ces problèmes : les règles musicales deviennent des contraintes que doivent vérifier les objets musicaux qu'elles lient.

La *programmation par contraintes* a pour objet non seulement la représentation de problèmes par des contraintes, mais aussi la résolution de ceux-ci par un ensemble de techniques algorithmiques efficaces. L'emploi de telles techniques est primordial vu la nature fortement combinatoire d'un CSP : si N est le nombre de variables contraintes et d la taille du plus grand de leur domaine, la complexité de résolution est au pire d^N, donc exponentielle.

On ne peut se contenter d'une résolution par un classique algorithme de *retour arrière chronologique (RAC),* comme on peut la trouver à la base de nombreux systèmes d'aide à la composition.

- M. Desainte-Catherine ([Desainte-Catherine 90]) a développé un langage déclaratif permettant de spécifier les relations de contraintes que doivent vérifier différents objets musicaux. La génération des combinaisons d'objets vérifiant ces relations s'appuie sur la stratégie de résolution du langage *Prolog,* c'est à dire le RAC.

- La démarche de F. Courtot présentée précédemment est tout à fait similaire puisqu'elle s'appuie aussi sur le langage *Prolog.*

- Chez K. Ebcioglu ([Ebcioglu 88]), la construction de l'harmonisation d'un choral est incrémentale et basée sur un mécanisme de retour arrière plus évolué. En cas de non conformité avec les règles d'écriture, toute étape élémentaire de construction sera remise en question d'une manière *intelligente.* Plutôt qu'effectuer une recherche arrière *chronologique* de la cause de l'échec, le principe est de directement détecter cette cause et ainsi, d'éviter avantageusement de suspecter toute décision antérieure étrangère à l'échec ([Berlandier 92], [Jegou 91]). Néanmoins, l'usage des contraintes est encore passif.

R. Ovans ([Ovans 92]) propose l'emploi de la programmation par contraintes pour pallier le manque d'efficacité d'OAC dont les algorithmes de résolution s'appuient fortement sur le RAC. L'idée est de gagner en efficacité en réduisant l'espace de recherche des solutions et en minimisant la probabilité de découvrir tardivement des impasses. Deux moyens principaux sont envisagés par Ovans.

1. *Avant l'énumération des solutions,* le principe est d'assurer un *prétraitement* du problème dans le but de prévenir l'occurrence de nombreux cas d'échec qu'on sait être inéluctables. Le *filtrage* des domaines de toutes les variables du problème permet la détection puis l'élimination de nombreuses valeurs ou combinaisons de valeurs qu'on sait être incompatibles avec l'ensemble des contraintes. Chaque domaine de variable ainsi épuré doit vérifier la propriété d'*arc-cohérence* ([Berlandier 92], [Jegou 91]).
2. *Pendant l'énumération des solutions,* le principe consiste à filtrer les domaines des variables non encore instanciées dans le but de supprimer toute valeur non compatible avec l'instanciation courante. Les techniques usant de ce mécanisme sont qualifiées de *look-ahead.* Dans le cas particulier du *forward-checking*, on ne filtre que les variables voisines de la dernière variable instanciée, alors que pour le *full-look-ahead,* toutes les variables non encore instanciées sont filtrées.

Ovans teste et confirme l'efficacité de son approche lors de la génération automatique de petits textes musicaux en contrepoint. Un tel exemple peut se contenter d'une assimilation des objets musicaux à des variables numériques de domaines finis. Néanmoins, il est peu probable qu'un telle représentation sommaire, sans structuration possible des objets, puisse convenir à un compositeur exigeant. D'où l'intérêt d'essayer de mêler intimement la richesse de la représentation des connaissances de Courtot à l'efficacité des techniques de programmation par contraintes proposées par Ovans.

3 Développer un OAC

3.1 Représenter les structures musicales

La première idée retenue est donc celle d'une représentation des structures musicales sous la forme de *types.* Suivant en cela la philosophie de B. Meyer ([Meyer 90]), nous proposons l'implantation informatique de ces types dans un formalisme à *objets*. Chaque objet musical devient un objet informatique dont on définit la structure par des *attributs* et le comportement par des *méthodes*. Ces objets peuvent être organisés en hiérarchies à l'intérieur desquelles chaque niveau hérite des caractéristiques structurelles et comportementales du niveau supérieur.

Le système de Courtot offre un certain nombre de caractéristiques assistant le compositeur dans sa tâche de modélisation. L'usage d'un langage à objets dépourvu d'interface est, par contre, réservé à l'informaticien. Au lieu de pouvoir spécifier graphiquement les relations entre types à l'aide d'opérateurs constructeurs de listes ou de taxonomies, il faut explicitement ajuster la structure des objets ou leur agencement pour tenir compte de ces relations.

Nous distinguons deux types d'opérateurs chez Courtot.

- *Les opérateurs syntaxiques* correspondent aux constructeurs de listes ou à la composition de types sans contrainte. L'utilisation de ce type d'opérateurs équivaut, dans le langage à objets, à la création de champs. Si un type T_1 est défini comme une liste d'éléments du type T_2, on créera un champ dans l'objet O_1 à même de recevoir une liste d'instances de l'objet O_2.

- *Les opérateurs sémantiques* correspondent à l'opérateur de composition de types avec contraintes (*prod*) ou aux opérateurs d'association (*equal* ou *interval*). Les relations qu'ils traduisent sont de nature intrinsèquement dynamiques.

Un exemple caractéristique d'une telle liaison est fournie par le type *note,* constitué par l'union des trois sous-types *nom, altération* et *octave.* Cette union ne se fait pas sans contraintes puisqu'il faut tenir compte des équivalences de hauteur des notes enharmoniques.
Chez Courtot, l'opérateur *prod* utilisé nécessite statiquement la donnée de tous les jeux de notes équivalentes.

L'opérateur *equal* permet, quant à lui, de lier deux types représentant des visions différentes d'un même objet. Un type *note* peut être ainsi lié à un type entier représentant la hauteur numérique de cette note. Chez Courtot, il faut encore donner statiquement les équivalences entre notes et valeurs numériques les représentant.

Nous pensons que la *programmation par contraintes* est apte à modéliser de telles relations de dépendance, parce qu'elle permet une liaison *dynamique* non directionnelle des quantités en relation. Une note est un *objet* constitué de cinq attributs mis en relations au moyen de *contraintes.*

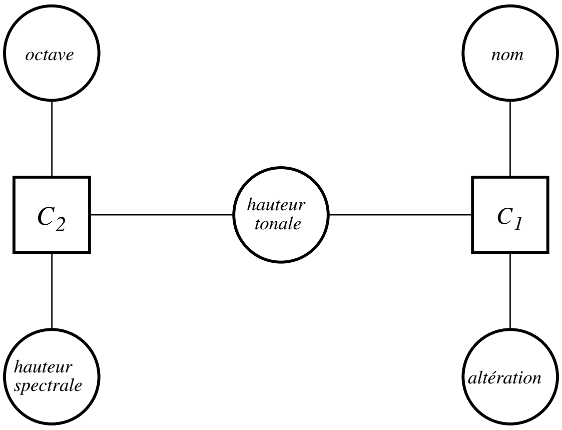
champ	domaine
nom	do, ré, mi, fa, sol, la, si
altération	double-bémol, bémol, bécarre, dièse, double-dièse
octave	-1…7
hauteur -tonale [a]	-1(*do bb*)…14(*si X*)
hauteur -spectrale [b]	-59(*do bb* -1)…52(*si X 7*)

[a] Équivalent numérique d'une note de nom et d'altération donnés sur une échelle de demi-tons.

[b] Idem, mais le numéro d'octave de la note intervient également dans le calcul de la hauteur .

Une relation algébrique C_1 lie les trois attributs *nom, altération* et *hauteur-tonale,* alors qu'une autre relation algébrique C_2 lie les attributs *hauteur-tonale, octave* et *hauteur-spectrale.*

FIGURE 1 : *Graphe biparti d'une note.*



L'aspect dynamique et relationnelle des contraintes implique que toute action (instanciation, réduction de domaine…) sur l'une quelconque des variables liées par une contrainte, a des répercussions automatiques et efficaces sur les autres, et ceci conformément aux contraintes les liant. Ce caractère *déductif* des contraintes est dû à l'emploi des méthodes d'*arc-cohérence* ou de *look-ahead.*

L’usage des contraintes peut être étendu de la simple modélisation des relations de dépendance entre attributs d’un même objet, à celle des relations entre objets. Un exemple caractéristique est la relation étroite existant entre un objet *intervalle* et les deux objets *note* le bornant. La modélisation de cette dépendance par une contrainte ouvre la voie à la modélisation de nombreuses règles d’écriture musicale où l’on considère tout autant les intervalles que les notes (construction des accords, intervalles prohibés, enchaînements mélodiques…).

Notre premier argument a donc été de montrer comment l’utilisation des contraintes pouvait apporter plus de souplesse, de dynamisme et d’efficacité à une représentation des structures musicales par objets fortement inspirée par celle de Courtot.

3.2 Représenter les règles musicales

La seconde étape est celle de la définition des règles que doivent vérifier les structures musicales. La modélisation de problèmes de composition musicale comme des CSP découle naturellement de la vision déclarative de tels problèmes : trouver une ou l’ensemble des combinaisons d’objets vérifiant le réseau de contraintes. Une règle musicale devient donc une contrainte mettant en relation plusieurs objets musicaux, par l’intermédiaire des attributs de ces objets.

Courtot propose un outil graphique permettant la saisie par le compositeur des relations que les structures musicales doivent vérifier . Là encore, faute d’interface, nous sommes plus accessibles au programmeur qu’au compositeur . Néanmoins, la richesse du langage de programmation par contraintes que nous utilisons [1] facilite la traduction des règles musicales à respecter . Ce langage s’appuie sur le modèle objet dans lequel nous représentons les structures musicales. Il offre principalement la possibilité de manipuler des variables contraintes de tout type (symbolique, entier , réel, ensembliste) et de les utiliser en association avec les objets, la disponibilité d’une bibliothèque de contraintes prédéfinies (relations d’ordre, opérations arithmétiques, contraintes symboliques [2]…) et un ensemble très puissant et paramétrable d’algorithmes de résolution.

L’expression d’une contrainte peut, soit utiliser une contrainte prédéfinie ou une composition de celles-ci, soit nécessiter la définition de nouveaux types de contraintes. Il est ainsi possible d’étendre le langage. Cette définition se fait au moyen de *schémas de contraintes* qui définissent les règles de propagation activables lors de toute modification des variables contraintes liées.

L’efficacité dans la recherche des solutions est obtenue par l’emploi d’algorithmes de propagation de contraintes comme l’ *arc-cohérence,* le *forward-checking* et le *full-look-ahead,* ainsi que par la possibilité de paramétrer les algorithmes d’énumération (critères de choix de la prochaine variable à instancier , de la prochaine valeur à choisir dans le domaine de celle-ci…). Toutes ces techniques permettent une réduction conséquente du temps de recherche des solutions.

Notre second argument a donc été de montrer le naturel et l’efficacité d’une représentation des règles de construction musicale par un formalisme de contraintes.

4 Résolution d’exercices d’harmonie à quatre voix

Nous avons développé à partir des idées exposées aux deux paragraphes précédents, un outil d’aide à la réalisation d’exercices d’harmonie à quatre voix. Même si l’intérêt musical de tels exercices reste scolaire, la méthode pour parvenir à leur résolution reste très représentative de nombreux problèmes de composition musicale. Elle mérite donc d’être explicitée afin de servir de modèle à d’autres développements.

---

[1] *Ilog-Solver* de *Ilog* ([Pecos 93a]). Cet outil s’appuie sur le langage *Le-Lisp* ([Le-Lisp 91]).

[2] Par exemple, la possibilité de contraindre le nombre de variables prenant une valeur donnée. Cette contrainte est très utile à l’intérieur d’un accord.

4.1 Modélisation

La réalisation d'un exercice d'harmonie à quatre voix du type *basse donnée*, consiste en la construction à partir de chaque note de la ligne de basse, des trois voix mélodiques supérieures. Cette construction se fait en respect des règles de l'harmonie ([Desportes 77], [Dubois 57]). Le choix des structures musicales tombe relativement sous le sens : les quatre lignes mélodiques sont des successions de notes sans rythme, formant verticalement des structures d'accords à quatre voix. Des intervalles mélodiques lient deux notes consécutives dans une même voix, alors que des intervalles harmoniques lient verticalement les différentes notes d'un même accord. La structure des accords est complétée par la présence d'attributs relatifs à leur degré, à leur état (fondamental ou l'un des renversements) et à leur type (majeur, mineur…). A chaque note est également attribué un degré. Des considérations d'analyse harmonique imposent enfin la présence d'une structure *tonalité* comprenant une note *tonique* et un mode (on ne considère pas les modulations). Toutes les contraintes modéliseront les relations complexes que doivent vérifier ces différents attributs ou objets dans un exercice correctement réalisé.

4.2 Réalisation

La première étape de la réalisation est celle de la définition des structures musicales (*note, intervalle, accord, tonalité…*) et des contraintes constructives qu'elles induisent. La plupart de ces dernières sont générales et dépassent le cadre de ces exercices. Moyennant quelques aménagements, elles pourraient être réutilisées dans d'autres contextes. C'est le cas pour la contrainte liant les différents champs d'un objet *note* ou pour la contrainte liant deux objets *note* à l'objet *intervalle* qu'ils bornent. L'expression de ces deux contraintes musicales ne requiert que l'utilisation de contraintes prédéfinies du langage. Schématiquement, la première de celles-ci se déclare ainsi :

```
; La contrainte prédéfinie ct-element permet la conversion d'une variable
; symbolique en variable numérique représentant son indice dans une liste.
; L'entier indice-n est contraint à être l'indice de nom-note dans la liste
; des noms de note : do=0 ré=1 …
(ct-element nom-note indice-n liste-noms-note)
; idem double-bémol=0 bémol=1 …
(ct-element alteration-note indice-a liste-alterations-note)
; exemple d'expression de la contrainte arithmétique C1 au moyen de
; contraintes arithmétiques prédéfinies
(ct-eq  (ct-mul 4 hauteur-tonale-note)
        (ct-add (ct-sub  (ct-add (ct-mul indice-n 7)
                                 (ct-mul indice-a 4))
                5)
        5))
; méthode similaire pour C2 …
```

Un autre aspect dépassant le cadre de ces exercices, concerne la notion de tonalité. On imagine qu'il y a un rapport de dépendance étroit entre d'une part la tonalité, les différentes notes de basse et leurs degrés, et d'autre part le type, le degré, l'état et les notes de chaque accord. Une famille de contraintes modélisent ces relations tonales. Notons que le caractère relationnel des contraintes rend également possible leur utilisation en analyse harmonique[1] : dans ce cas, on renverse alors en quelque sorte le processus de construction. Par exemple, au lieu de construire les notes d'un accord à partir de son type, on peut déterminer le type à partir des notes, puis toutes les tonalités auxquelles cet accord peut appartenir…

---

[1] On pourrait opposer cette approche à celle de R. Rowe qui emploie un réseau neuronal pour déterminer la tonalité d'un morceau ([Rowe 92], [Scarborough 89]). Les contraintes ont l'avantage de coller de très près aux méthodes mentales qu'aurait employées un élève confronté à ce problème.

La seconde étape de la réalisation consiste en la traduction sous la forme de contraintes, des règles de l'harmonie gouvernant la résolution des exercices. L'effet de certaines de ces contraintes peut être de restreindre la liberté des objets sur lesquels elles s'appliquent. Parmi celles-ci, citons de manière non exhaustive les contraintes imposant les tessitures de chaque voix, évitant les croisements entre voix, évitant une distance supérieure à l'octave entre voix, interdisant les successions d'intervalles harmoniques prohibés, limitant les types d'intervalles mélodiques admissibles entre deux notes consécutives, traitant le problème des doublures de notes, évitant le mouvement harmonique direct… D'autres contraintes agissent plutôt comme des règles d'inférence se déclenchant dès que les prémisses de leurs schémas sont vérifiées : mouvements de notes obligés, traitement des unissons… Comment s'expriment dans le formalisme de *Ilog-Solver* de tels schémas, par exemple dans le cas où l'on veut contraindre le mouvement de la note *sensible* ?

```
 (when  ; Quand on détecte la sensible au soprano du premier accord
        ; elle doit monter à la tonique du second accord.
test    ; Quand le premier accord est un accord du cinquième degré,
        (and (eqn d-accord-1 5)
             ; et quand le second accord contient la tonique (degré 1, 4 ou 6)
             (or (eqn d-accord-2 1) (eqn d-accord-2 4) (eqn d-accord-2 6))
             ; et quand le soprano du premier accord est la sensible
             (eqn d-soprano-1 7)
assert  ; alors le soprano du second accord sera la tonique.
        (d-soprano-2 = 1))
; d'autres schémas pour les autres voix …
```

La possibilité d'inclure n'importe quelle expression booléenne à l'intérieur des prémisses d'un schéma permet le traitement des cas particuliers aux règles, si fréquents en musique. Par exemple, pas de quinte directe entre parties extrêmes sauf sur l'accord du I et du V si la partie supérieure procède par mouvement conjoint et sauf sur les autres degrés si… Tous les caprices sont permis.

Autre remarque : le schéma précédent montre que l'utilisation des degrés permet de se situer à un niveau d'abstraction plus élevé que celui des notes réelles. Il est alors possible d'exprimer toute contrainte sur les enchaînements de degrés et ce faisant, de traiter les cadences. Tout ce traitement est indépendant de la tonalité.

4.3 Résolution

La définition des structures musicales et des règles que celles-ci doivent vérifier, achève l'énoncé déclaratif du problème : la phase de précomposition est spécifiée. Comme le remarque Courtot, cette description est la plupart du temps *sur-générale* si bien que le nombre de combinaisons des valeurs de toutes les variables ou objets vérifiant le réseau de contraintes est généralement important. On qualifie de *sous-contraint* un tel problème.

Se pose le problème du paramétrage de l'algorithme d'énumération des solutions, notamment en ce qui concerne le choix de l'ordre d'instanciation des variables. L'aspect relationnelle des contraintes laisse penser que cet ordre n'a pas d'importance. Certes, quel que soit celui-ci, une solution sera toujours trouvée, mais les variations du temps d'exécution peuvent être énormes. Pour minimiser ce temps, il faut se laisser guider par la topologie du réseau de contraintes. Celle-ci est liée à la manière dont on a exprimé les contraintes. On entend par là, la manière dont les variables s'agencent quasi *géographiquement* les unes par rapport aux autres au moyen des contraintes. Ainsi, vaut-il mieux éviter d'instancier les notes d'un accord avant les intervalles harmoniques le formant, car la contrainte de construction des accords raisonne justement sur les intervalles plutôt que sur les notes. Similairement, on préférera déterminer d'abord la tonalité de l'exercice avant d'essayer d'instancier les degrés des accords dont ils dépendent…

De plus, certains schémas de contraintes sont clairement directionnels[1] si bien que de fait, ils imposent un ordre d'instanciation indispensable à leur efficacité.

L'idée est donc d'éviter d'avoir à faire un tel choix de manière non réfléchie, car la remise en cause tardive des conséquences de ce choix est coûteuse. On peut dire que l'ordre d'instanciation retenu dépend de l'expression des contraintes et de leur chronologie optimale d'activation. Cet ordre est finalement assez naturel puisque il correspond à celui d'un élève résolvant l'exercice sur le papier .

Une fois le choix de la variable à instancier effectué, peut se poser le problème du choix de l'ordre dans lequel vont être essayées les différentes valeurs de son domaine. Un peu à la manière du système d'Ebcioglu, *Ilog-Solver* permet la définition d'heuristiques propres à l'application et fixant cet ordre. Dans le cas des exercices d'harmonie, nous n'avons pas pour l'instant exploité cette possibilité. Aussi, les domaines des variables sont ils examinés dans l'ordre croissant de leur déclaration.

Le tableau suivant regroupe un certain nombre de résultats obtenus lors de la réalisation d'enchaînements de un à douze accords.

enchaînements	nombre d'accords	nombre de solutions	nombre de variables	nombre d'échecs[a]	nombre d'événements de propagation (x 10^3)[b]	temps de calcul de toutes les solutions	temps de calcul de la première solution[c]
sol_2	1	14	131	253	99,6	24,2 s	2,7 s
do_2		7		144	58,5	12,6 s	2,1 s
la_1		15		221	94,6	22,9 s	3,2 s
$ré_2$		17		265	98,8	24,2 s	2,6 s
sol_2-do_2	2	3	290	452	239,2	57,6 s	19,8 s
la_1-$ré_2$		42		1811	885,7	3 mn 50	8,6 s
	3	8	449	1725	911,2	4 mn 07	1 mn 03
	4	20	608	4473	2461	10 mn 33	1 mn 48
	5	12	767	3270	1839	7 mn 52	2 mn 37
	6	56	926	15472	8516	36 mn 43	2 mn 12
	7	118	1085	32682	17967	77 mn 36	2 mn 21
	8	36	1244	14186	8301	36 mn 05	3 mn 25
	12	>52	1880	?	?	?	3 mn 08
	16	>199	2516	?	?	?	4 mn 03

[a] Pendant le processus d'énumération, un échec survient chaque fois que le domaine d'une variable devient vide. Un *retour arrière* se produit alors.

[b] Ces événements correspondent à toute modification touchant une variable (affectation de valeur , retrait d'éléments du domaine, modification des bornes du domaine) et provoquée par l'activation d'une contrainte.

[c] Sur Sparc 10.

4.4 Évaluation

Avant de comparer plus précisément notre approche à celle de C.P . Tsang, apportons quelques commentaires généraux sur les chiffres précédents.

- Le nombre de retours arrière est relativement peu important par rapport au nombre de variables et à la taille des domaines, preuve de l'efficacité des méthodes de propa-

---

[1] Pour des contraintes traduisant certaines règles complexes (problème des doublures…), il n'est pas toujours facile d'exprimer les schémas inverses qui auraient rendu la contrainte plus efficace et clairement non *directionnelle.*

gation de contraintes (*arc-cohérence, full-look-ahead*) utilisées par *Ilog-Solver*. La réduction très importante des domaines par filtrage est compréhensible, compte tenu du grand nombre d'événements de propagation (EP). La cause de ce grand nombre d'EP réside dans la nature fortement *connexe* du graphe de contraintes : les variables sont intimement liées entre elles au moyen de contraintes. Le cas d'une note est significatif : celle-ci dépend des notes précédente et suivante, de l'accord auquel elle appartient, des autres notes de l'accord, de la tonalité… Signalons qu'environ 20 % des EP sont provoqués par des réductions de domaines dûes à la contrainte *ct-element.* L'emploi de celle-ci permet la manipulation utile mais hélas coûteuse de domaines symboliques.

- En l'absence d'heuristiques de choix de valeurs dans le domaine des variables, ceux-ci sont passés en revue dans l'ordre croissant. Aussi, la première solution trouvée correspond-elle aux positions les plus basses des accords.

- Du fait d'une modélisation riche et dépassant le cadre de la résolution de simples exercices d'harmonie, le nombre de variables contraintes est très important. Or, la complexité de résolution d'un CSP est liée à ce nombre de variables et à la taille de leur domaine. Un bon principe sera de choisir une représentation minimale, épurée mais non réductrice, qui dépendra du problème posé. Dans cet état d'esprit, nous comptons développer une application dédiée à la génération de *canons obligés.* Notons qu'une autre conséquence d'une représentation trop générale est une occupation mémoire importante. Dans notre cas, la génération de douze accords requiert une zone de travail de sept méga-octets.

FIGURE 2 : *Enchaînement de seize accords en do majeur.*

4.5 Comparaison

C.P. Tsang ([Tsang 91]) utilise la programmation logique avec contraintes (CLP) pour harmoniser à quatre voix une ligne mélodique donnée. Ce type de programmation permet la prise en compte de contraintes au sein du mécanisme d'unification à la base de la programmation logique ([Cohen 90]). L'argument de Tsang est la parfaite adéquation entre ce formalisme et l'expression déclarative d'un problème d'harmonisation. Notons que le problème auquel s'attaque Tsang est le symétrique du nôtre.

Puisque construit sur *Prolog,* un outil de CLP offre théoriquement un avantage sur un outil de résolution de CSP : les capacités du premier englobent et surpassent celles du second. Tsang argue que le traitement non déterministe rendu possible par l'outil de CLP rend *dynamique* la spécification d'un CSP, ce qui est souhaitable dans un problème d'harmonisation. Le traitement d'alternatives entre contraintes est alors possible. Si l'idée semble bonne, l'exemple que Tsang produit pour justifier son argumentation laisse penser que la nécessité d'un contrôle non déterministe est plutôt dû au caractère rudimentaire de l'outil qu'il utilise, notamment en ce qui concerne la difficulté à exprimer des contraintes conditionnelles.

En *Ilog-Solver*, bâti lui sur *Le-Lisp,* il est tout à fait possible d'exprimer directement dans le langage de contraintes, la règle interdisant l'usage du second renversement du cinquième degré, sauf entre un quatrième et un sixième degré. Cette expression se fait sans avoir recours à la programmation non déterministe. Notons qu'*Ilog-Solver* offre, si besoin est, un

PostScript Lang. Ref. Man., 2nd Ed., H.2.4 says EPS must not call setpagedevice

certain nombre de primitives de contrôle non déterministe (notamment des *conjonctions* ou *disjonctions* d'expressions non déterministes) permettant de simuler une programmation *Prolog* encapsulant la pose des contraintes . Toutefois, dans le cadre du problème que nous nous sommes fixé, nous n'avons pas eu besoin de recourir à cette possibilité, preuve de la richesse d'expression du langage *Ilog-Solver* .

Par ailleurs, l'utilisation d'un outil de CLP ne permet pas de s'appuyer de manière naturelle sur une modélisation par objets des structures musicales, faculté qui nous semble pourtant fondamentale. Chez T sang, la représentation des connaissances est de ce fait sommaire. Les notes y sont encore codées de manière numérique, les intervalles étant des différences entre ces nombres. Malgré cette simplicité de représentation, T sang reconnaît les mauvaises performances de son système : lenteur d'exécution (cinq minutes pour *une* harmonisation de onze accords) et occupation mémoire gigantesque (70 méga-octets). Pour traiter un problème similaire, *Ilog-Solver* requiert dix foix moins de mémoire. Quant aux temps d'exécution, nous faisons au moins aussi bien, avec une représentation des connaissances bien plus riche. A égalité de représentation des connaissances, les temps d'exécution seraient incomparables.

5 Affiner les résultats

5.1 Deux types de critères d'appréciation

Le compositeur ne se contentera pas d'une génération exhaustive de toutes les solutions vérifiant les contraintes de construction qu'il s'est fixées. Il voudra dépasser ce stade en sélectionnant parmi ces solutions, celles répondant à certains critères d'appréciation artistique. Deux cas de figure peuvent généralement se présenter .

1. Ces critères correspondent à des conditions supplémentaires que les solutions devront vérifier [1]. La prise en compte de celles-ci s'effectue par la pose de nouvelles contraintes sur les structures musicales. Ainsi, l'étape de précomposition s'affine en devenant plus sélective.

2. Ces critères correspondent à des *baromètres* jaugeant l'intérêt musical des solutions. Dans ce cas, il ne s'agit plus de contraindre toute solution à les vérifier , mais de chercher à les optimiser . Plus une solution maximisera ces critères d'évaluation, plus elle correspondra aux souhaits du compositeur .

   Pour réaliser cette optimisation, nous choisissons d'utiliser l'algorithme de *branch and bound* fourni par l'outil *Ilog-Solver* . Le principe d'action de cet algorithme est de chercher à minimiser, tout au long de la phase d'énumération des solutions, une variable *coût* donnée. A tout instant, toute solution partielle dont le coût excéderait celui de la meilleure solution complète déjà trouvée, sera abandonnée. Il en résulte un gain en efficacité puisque de nombreuses branches de l'arbre de recherche des solutions ne seront pas explorées.

5.2 Application aux exercices d'harmonie

La réalisation correcte d'un exercice d'harmonie dépasse le simple respect de règles musicales. Une solution ne doit pas être exempte d'intérêt musical. Aussi, faudra-t-il accorder beaucoup d'importance aux mouvements mélodiques des différentes voix. Il est possible d'influer sur ces mouvements par l'intermédiaire de considérations sur la nature des intervalles mélodiques entre notes voisines au sein de chaque voix.

---

[1] Dans le cadre des exercices d'harmonie, on voudra par exemple imposer la présence de la tonique au soprano lors des cadences parfaites ou bien préciser la position de l'accord initial…

On peut songer par exemple, pour la voix supérieure, à interdire les unissons mélodiques, à éviter autant que possible les grands sauts d'intervalles et les changements exagérés de sens de variation. On cherchera par contre, à maintenir les voix intermédiaires les plus conjointes possibles. Le respect de telles exigences se fait soit par la pose de contraintes supplémentaires, soit par le biais de l'algorithme de *branch and bound.* La figure suivante illustre l'amélioration de fil en aiguille des solutions, par rapport aux critères souhaités.

FIGURE 3 : *Amélioration itérative de la conformité des solutions aux critères.*

	6	6	6
nombre de changements de sens de direction au soprano [a]	3	2	1
nombre d'intervalles conjoints au soprano [b]	2	2	3
nombre d'unissons mélodiques à l'alto et au ténor [c]	2	2	3

a. On cherche à minimiser ce nombre.

b. En fait, on cherche à minimiser la somme des étendues en demi-tons de tous les intervalles mélodiques du soprano. On a de plus interdit par une série de contraintes, la présence d'unissons mélodiques dans cette même voix.

c. On cherche à maximiser ce nombre.

Généralement, il n'est pas aussi aisé de définir un ensemble de critères dont le respect pourrait garantir à coup sûr , l'intérêt musical du résultat. Les critères entretiennent en effet, des relations complexes qui dépassent souvent le cadre d'une simple optimisation de leur somme pondérée. On peut dire que le respect indépendant de chacun d'entre eux n'est ni une condition nécessaire, ni une condition suffisante à l'obtention d'un résultat musicalement intéressant. D'une part, certains critères peuvent en effet, être antinomiques : la satisfaction de certains d'entre eux entraîne *a fortiori* la non satisfaction d'autres [1]. D'autre part, des phénomènes de compensation entre critères existent : la suffisante satisfaction d'un groupe d'entre eux autorise la non satisfaction d'un autre groupe.

La difficulté réside alors dans la fusion de cet ensemble de critères liés par des relations complexes, dans un unique critère général que l'OAC pourra optimiser . Dans le formalisme d'*Ilog-Solver* , nous pensons modéliser ces relations par un ensemble de contraintes liant les différentes variables *critères* . Il est possible de définir des sommes pondérées de celles-ci tout autant que les relations conditionnelles déclarant la non pertinence de tel ou tel critère en fonction de circonstances particulières.

6 Conclusion

6.1 Bilan

Dans le cadre d'une vision déclarative des problèmes de composition musicale, un outil d'aide doit permettre non seulement la représentation naturelle et non réductrice des structures musicales que le compositeur désire manipuler , mais aussi la définition et l'application efficace des règles d'organisation que ces structures doivent vérifier .

La première étape de notre argumentation a été de prouver qu'un OAC reposant sur un formalisme mêlant programmation par objets et programmation par contraintes, répond à ce

---

[1] Par exemple, la recherche du mouvement contraire entre le soprano et la basse aurait tendance à augmenter le nombre de changements de sens de direction de la voix supérieure (et inversement).

double objectif. D'une part, la richesse d'une représentation par objets alliée à l'aspect relationnel des contraintes, autorisent une fine modélisation des structures musicales. D'autre part, les règles musicales que doivent vérifier ces structures deviennent avantageusement des contraintes. L'intérêt est de disposer de techniques de consistance et d'un algorithme d'énumération paramétrable et efficace dont l'usage entraîne une réduction notable de la complexité du problème.

Notre seconde étape a été de mettre en œuvre une telle méthodologie de construction dans le cadre de la réalisation assistée d'exercices d'harmonie. Le but recherché, plus que le résultat musical, fut d'expliciter les phases de déclaration et de résolution d'un problème concret, en donnant une idée des formalismes ou techniques informatiques employés.

L'ultime étape de notre argumentation a permis de dépasser le stade d'une génération exhaustive et peu musicale d'objets se conformant à des règles. L'emploi d'un algorithme d'optimisation permet la prise en compte d'heuristiques garantissant un meilleur intérêt musical du résultat obtenu. Pousser plus loin dans cette voie pourrait transformer l'OAC en programme de composition automatique.

6.2 Limitations et perspectives

La première limitation de notre approche est sa non accessibilité au compositeur non familier de programmation. Ce fait est dû à l'absence d'interface, en particulier d'outils d'aide à la déclaration des structures ou contraintes musicales. Là encore, le système de Courtot pourrait être notre modèle pour sa facilité d'emploi. Inversement, l'utilisation des contraintes que nous proposons apporterait à ce système plus d'efficacité.

Le manque de souplesse des contraintes est cause de la deuxième limitation de notre système. Celui-ci échoue face à un problème sur-contraint : aucune combinaison d'objets ne vérifie les règles musicales trop contraignantes que l'on s'est fixées. Un compositeur confronté à cette même situation serait capable de réagir en relâchant certaines contraintes jugées les moins fondamentales, de manière à obtenir une solution approchée. Une évolution souhaitable de notre système serait de permettre une déclaration *hiérarchique* des contraintes : certaines seront *requises*, d'autres se verront attribuées un *coefficient d'importance* indiquant combien il est important qu'elles soient satisfaites. L'algorithme de résolution devra alors tâcher de satisfaire d'une manière opportuniste les contraintes les plus *faibles*, tout en veillant au scrupuleux respect de celles requises ou des plus *dures* d'entre elles.

Références

[Arveiller 85] J. Arveiller, Art et informatique, *Encyclopædia Universalis,* vol. 2, pp. 1 164-1 168.

[Berlandier 92] P. Berlandier, *Etudes de Mécanismes d'Interprétation de Contraintes et de leur Intégration dans un système à base de connaissances,* Thèse de doctorat, Université de Nice, 1992.

[Billard 85] P. Billard, Musique, *Encyclopædia Universalis,* vol. 12, pp. 820-823.

[Blevis 92] B. Blevis, Motivations, Sources, and Initial Design Ideas for CALM : A Composition Analysis/ Generation Language for Music, *Understanding Music with AI : Perspectives on Music Cognition,* AAAI Press, Menlo Park, California, 1992.

[Charbonnier 85] G. Charbonnier, Art et mathématique, *Encyclopædia Universalis*, vol. 2, pp. 1795-1801.

[Cohen 90] J. Cohen, Constraint Logic Programming languages, *Communications of the ACM,* vol. 33, n° 7, Juillet 1990.

[Courtot 90] F. Courtot, Représentation évolutive pour l'aide à la composition, *Actes du colloque structures musicales et assistance informatique,* Laboratoire musique et informatique de Marseille (MIM), Marseille, 1990.

[Courtot 92a] F. Courtot, Logical Representation and Induction for Computer Assisted Composition, *Understanding Music with AI : Perspectives on Music Cognition,* AAAI Press, Menlo Park, California, 1992.

[Courtot 92b] F. Courtot, *CARLA : acquisition et induction sur le matériau compositionnel,* Thèse de doctorat, Université de Rennes I, 1992.

[Desainte-Catherine 90] M. Desainte-Catherine, Un langage déclaratif pour la composition musicale*, Actes du colloque musique et assistance informatique*, Laboratoire musique et informatique de Marseille (MIM), Marseille, 1990.

[Desportes 77] Y. Desportes, *Traité d'harmonie en vingt leçons*, Billaudot, Paris, 1977.

[Dubois 57] T. Dubois, *Traité d'harmonie : théorie et pratique*, Heugel & C^{ie}, Paris, 1957.

[Ebcioglu 88] K. Ebcioglu, An Expert System for Harmonizing Four-part Chorales, *Computer Music Journal,* vol. 12, n° 3, pp. 43-51, Fall 1988.

[Hoffstader 82] D. Hoffstader, Forme, force et fantaisie dans la musique de Frédéric François Chopin, *Mathémagie, en quête de l'essence de l'esprit et du sens,* Inter Éditions, Paris, 1988.

[Jegou 91] P. Jegou, *Contribution à l'étude des problèmes de satisfaction de contraintes : algorithmes de propagation et de résolution, propagation de contraintes dans les réseaux dynamiques*, Thèse de doctorat, Université de Montpellier, 1991.

[Le-Lisp 91] *Le-Lisp version 15.25, manuel de référence,* ILOG, Gentilly, 1991.

[Loy 88] G. Loy, Composing with Computers - a Survey of Some Compositional Formalisms and Music Programming Languages*, Current Directions in Computer Music Research,* MIT Press, Cambridge, 1989.

[Machover 81] T. Machover, Le compositeur et l'ordinateur : quelques réflexions, *Le compositeur et l'ordinateur*, IRCAM, Paris, 1981.

[Machover 85] T. Machover, Le concept de recherche musicale, *La recherche musicale : quoi ? quand ? comment ?* Christian Bourgois éditeur, IRCAM, Paris, 1985.

[Molino 88] J. Molino, Musique et machine*, Actes du colloque : structures musicales et assistance informatique*, Laboratoire musique et informatique de Marseille (MIM), Marseille, 1988.

[Meyer 90] B. Meyer, *Conception et programmation par objets : pour du logiciel de qualité,* Inter éditions, Paris, 1990.

[Ovans 92] R. Ovans, *Efficient Music Composition via Consistency Techniques,* Center for Systems Science, Simon Fraser University, Burnaby, Canada, 1992.

[Pecos 93a] *Pecos*[1] *version 1.2, manuel de référence*, ILOG, Gentilly, 1993.

[Pecos 93b] *Pecos version 1.2, manuel d'exemples*, ILOG, Gentilly, 1993.

[Philippot 85] M. Philippot, Composition musicale*, Encyclopædia Universalis,* vol. 5, pp. 232-235.

[Rowe 92] R. Rowe, Machine Listening and Composing with Cypher*, Computer Music Journal,* vol. 16, n° 1, pp. 43-63, 1992.

[Scarborough 89] D.L. Scarborough, B.O. Miller, J.A. Jones, Connectionist Models for Tonal Analysis, *Computer Music Journal,* vol. 13, n° 3, pp. 49-55, 1989.

[Schœnberg 67] A. Schœnberg, *Fondements de la composition musicale*, J.-C. Lattès, Paris, 1987.

[Tsang 91] C.P. Tsang, Harmonizing Music as a Discipline of Constraint Logic Programming, *Proceedings of the International Computer Music Conference,* Montréal, Canada, 1991.

---

[1] Pecos est l'ancien nom de l'outil *Ilog-Solver*.