Janvier 1997 – Journées Francophones des Langages Applicatifs – JFLA97

# Conception de problèmes par objets et contraintes

Pierre Roy & François Pachet

*Laforia-IBP, Université Paris 6, Boîte 169*
*4, place Jussieu,*
*75252 Paris Cedex*
*Tel : (33) 01 44 27 70 04*
*Fax : (33) 01 44 27 70 00*
*roy{pachet}@laforia.ibp.fr*

**Résumé**

Plusieurs systèmes proposent l'intégration d'algorithmes de satisfaction de contraintes au sein de langages à objets, mais le problème de la conception à l'aide de tels systèmes reste peu abordé. Nous nous intéressons ici à la conception de problèmes mettant en œuvre des objets composites et des contraintes portant à la fois sur des objets composites et sur des composants. Nous identifions deux approches radicalement différentes sur ce type de problèmes. Dans la première approche, les objets composites sont définis à l'aide de contraintes spécifiées dans les classes, et les objets se "remplissent" au fur et à mesure de la résolution. Dans la seconde, les objets sont systématiquement instanciés, et les phases d'instanciation et de résolution sont distinguées. Nous comparons ces deux approches et montrons comment combiner leurs avantages respectifs, à la fois en efficacité, et du point de vue de la réutilisation de librairies de classes. Nous illustrons ce résultat sur un problème difficile, l'harmonisation automatique, qui a été résolu avec les deux méthodes.

## 1. Introduction

La programmation par contraintes est un outil puissant pour la définition et la résolution de problèmes combinatoires. Les premiers résultats importants portèrent sur l'aspect purement algorithmique de la satisfaction de contraintes ([21], [18]) ; on s'est ensuite intéressé à son intégration dans des langages de programmation.

## 1.1. Les langages de programmation par contraintes

Les langages de programmation logique sont les premiers auxquels les mécanismes de satisfaction de contraintes ont été intégrés. L'extension de la programmation logique aux contraintes lui donne la capacité de calculer sur des domaines spécifiques (nombres réels, entiers, domaines finis), et constitue le paradigme de la programmation logique avec contraintes (CLP). Parmi les nombreuses réalisations de CLP, on peut citer Prolog III [9], CHIP [32] ou CLP (R) [14].

L'intégration de ces mécanismes dans un langage à objets repose sur une vision radicalement différente. En effet, les classes peuvent être vues comme une représentation intentionnelle des domaines et leurs instances comme des valeurs potentielles pour les variables contraintes. Dans un langage de programmation par objets (i.e. avec des méthodes associées aux classes), l'interface d'une classe fournit de plus un langage pour exprimer les contraintes. C'est dans ce cadre précis que nous nous situons.

## 1.2. La propagation locale

Le premier système intégrant des contraintes dans un langage de programmation par objets est ThingLab [5]. Les contraintes, dans ThingLab, sont vues comme un moyen de définir un "état stable" pour un système d'objets. ThingLab est alors chargé de rétablir la stabilité du système après une perturbation. Cette stabilisation met en œuvre une technique dite de propagation locale, dans laquelle chaque contrainte réagit à la perturbation en proposant de nouvelles valeurs, et ce jusqu'à l'obtention d'un nouvel état stable. Ce modèle est particulièrement adapté à la construction d'interfaces graphiques, dans lesquelles les contraintes définissent les relations entre objets graphiques. Un convertisseur dynamique entre degrés Fahrenheit et Celsius fournit un bon exemple : l'équation de conversion définit une contrainte qui est activée si l'utilisateur change la valeur de la jauge des degrés Celsius, afin de mettre à jour celle des degrés Fahrenheit.

Plusieurs systèmes ont suivi la lignée de ThingLab comme Kaleidoscope [11], [12], [20] ou Rollit [15], pour la construction interactive d'interfaces. Ces techniques sont également utilisées dans le domaine de la CAO [19].

## 1.3. La résolution de problèmes

La résolution de problèmes combinatoires définis avec des contraintes (CSP, pour constraint satisfaction problem) relève d'une approche très différente. La problématique n'est pas ici de maintenir la stabilité d'un système, mais de trouver une solution à un problème donné. La principale différence par rapport au modèle précédent est le rôle du système de résolution : au lieu de rétablir la stabilité d'un système après perturbation, il s'agit ici de calculer un ensemble cohérent de valeurs pour les variables, à partir des contraintes du problème. Après résolution, chacune des variables a reçu une valeur, appartenant à son domaine, de façon à ce que toutes les contraintes du problème soient respectées.

La puissance expressive des CSP, alliée à l'efficacité des algorithmes existants, permet de résoudre de nombreux problèmes comme l'allocation de ressources (e.g. gestion de ressources humaines), la gestion d'emploi du temps ou encore l'ordonnancement (e.g. réseaux de transports). C'est cette approche, historiquement moins étudiée par la communauté de la programmation par objets, que nous suivons. Plus précisément, nous abordons les problèmes dans lesquels les variables ont un domaine fini (listes d'objets, intervalles entiers etc.).

## 1.4. Satisfaction de contraintes et langages à objets

On compte de nombreuses réalisations de systèmes de satisfaction de contraintes dans des langages à objets, comme Ilog Solver [28], qui fournit des mécanismes de résolution de CSPs à domaines finis. Ce système, qui utilise un algorithme de résolution très efficace, a été employé dans de nombreuses applications industrielles d'envergure.

Parallèlement, plusieurs systèmes sont conçus comme des extensions de divers langages à objets : LAURE [7], qui constitue une implémentation extrêmement efficace des mécanismes de satisfaction de contraintes, COOL [1], intégré à l'environnement KEE, ainsi que Prose [3], bâti sur Smeci (une extension objets de Lisp). Concernant COOL, ses méthodes de résolution ne sont pas modifiables en fonction du problème à résoudre. Quant à Prose, ce n'est pas à proprement parler un système à objets, mais plutôt, comme le dit son auteur, un système aisément implémentable dans un langage à objets. Enfin, le système [13], en dotant le langage Tropes de capacités de satisfaction de contraintes à travers Pecos (la version Lisp d'Ilog Solver), illustre une autre vision de l'intégration des contraintes et des objets : l'intégration se fait via une *interface* entre les deux langages (Tropes et Pecos dans ce cas).

Nous nous intéressons à la réutilisation de librairies de classes pour la conception de problèmes de contraintes. Le choix de Smalltalk comme langage se justifie par la grande quantité de composants disponibles et par leur qualité. L'approche employée par Gensel ne nous convient pas car elle oblige à une double représentation des contraintes et des variables et est donc inadaptée à des systèmes nécessitant une utilisation intensive des contraintes. Il n'existe pas de telle librairie en Smalltalk, les systèmes basés sur la propagation locale tels que la librairie OTI Constraint Solver [6] n'étant pas adaptés à la résolution de problèmes combinatoires. Nous avons donc conçu notre propre système de satisfaction de contraintes à domaines finis : BackTalk [25], [29]. BackTalk se présente sous la forme d'une librairie de classes, représentant les concepts principaux de la satisfaction de contraintes : domaines, variables contraintes, contraintes, problèmes et algorithmes. À l'image du système de backtracking de [17], cette librairie à été programmée sans modification de la machine virtuelle, assurant ainsi sa portabilité. L'architecture de BackTalk est proche de celle de YAFCRS [16], mais s'en différencie par la présence d'une hiérarchie d'algorithmes, qui peuvent être employés indifféremment pour résoudre un problème donné. De plus, une représentation efficace des domaines finis, alliée à l'utilisation d'algorithmes performants, rend le système suffisamment efficace pour résoudre des problèmes difficiles (e.g. emploi du temps hospitalier, harmonisation musicale). Enfin, les domaines des variables contraintes sont susceptibles de contenir des objets quelconques, ce qui autorise la définition de problèmes impliquant des structures de données arbitrairement complexes.

Nous nous plaçons donc dans un cadre de travail double. D'une part les objets au sens de la programmation par objets, qui constituent une représentation a priori du monde modélisé, et que nous cherchons à réutiliser. D'autre part les contraintes qui vont permettre de spécifier un problème combinatoire à domaines finis portant sur ces objets.

Ce papier aborde le problème de la conception d'applications dans ce cadre précis. Sur un problème simple de géométrie, nous comparons deux solutions standards, et étudions leurs avantages et inconvénients (section 2). Nous proposons ensuite une démarche de résolution permettant de combiner leurs avantages (section 2.3). Enfin (section 3), nous appliquons notre démarche à un problème complexe, l'harmonisation musicale, et montrons comment cette démarche améliore très sensiblement la résolution, ainsi que la facilité de définition du problème et la réutilisation de classes ayant été créées dans un but différent.

## 2. Concevoir des systèmes de contraintes avec des objets

La définition d'un problème combinatoire dans le formalisme des CSP est une opération délicate. Des problèmes aussi simples que celui des 8 reines, ou le cryptogramme send+more=money, admettent plusieurs représentations, équivalentes quant à leurs solutions, mais fort différents en termes de nombre et nature des variables et des contraintes, et d'efficacité de la résolution. Dans notre cadre, à cette difficulté de définition des variables et des contraintes, s'ajoute le problème de séparer ce qui relève des mécanismes purement objets – instanciation, méthodes –, de ce qui relève des contraintes.
Afin d'illustrer cette difficulté, nous présentons deux formulations opposées pour un même problème de géométrie. Ces deux conceptions sont extrémistes : l'une n'utilise que les mécanismes des contraintes, l'autre s'appuie largement sur l'utilisation de méthodes, et l'instanciation. Voici l'énoncé de notre problème de référence :

(P) trouver toutes les paires de quadrilatères non triviaux satisfaisant les contraintes suivantes :

1. les sommets ont des coordonnées entières dans $\{1, \ldots, n\}$,
2. aucun sommet ne se trouve sur la première bissectrice,
3. les deux quadrilatères sont des rectangles "posés à plat",
4. les deux quadrilatères ne se rencontrent pas.

La Figure 1 donne une illustration pour n=10.

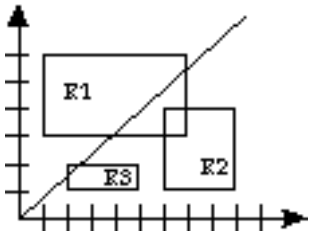


Figure 1: Ici n=10, les paires {R1, R3} et {R2, R3} sont des solutions correctes pour (P), tandis que {R1, R2} n'est pas une solution.

## 2.1. Deux formulations du problème sous la forme de CSP

Pour formuler ce problème dans le formalisme des CSPs, il faut définir les variables, puis les contraintes liant ces variables entre elles.

1) Formulation avec uniquement des contraintes
La première solution est de ne considérer, comme variables contraintes, que les points sommets des quadrilatères. On définit ainsi 8 variables contraintes $a, b, c, d$ et $a', b', c'$, et $d'$, correspondant aux sommets des deux quadrilatères. La formulation des relations (1) à (4) se fera par des contraintes impliquant ces variables. Par exemple, la relation (4) sera définie par la contrainte suivante :

$x(b) < x(a')$ "R1 est à gauche de R2"
*or* $x(a) > x(b')$ "R1 est à droite de R2"
*or* $y(c) > y(a')$ "R1 est au dessus de R2"
*or* $y(a) < y(c')$ "R1 est au dessous de R2"

(avec x et y les fonctions coordonnées d'un point, voir figure 2).

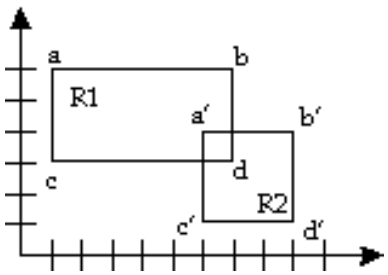


Figure 2: Les rectangles, représentés par leur sommets.

Dans cette approche, le problème est défini par 8 variables ayant un domaine de cardinalité $(n^2 - n)$, et 9 contraintes. Les relations (1) et (2) sont représentées dans les domaines ; (3) est exprimée par 8 contraintes binaires, et (4) par une contrainte globale (i.e. impliquant les 8 variables).

2) Formulation avec des contraintes et des objets
La formulation précédente n'utilise pas les structures d'objets mentionnées dans l'énoncé. Puisque le problème est exprimé à la fois en termes de points et de rectangles, il est naturel de définir des variables rectangles (i.e. dont le domaine contient des rectangles) et des variables points. L'intérêt majeur de cette formulation est de simplifier la définition des contraintes portant sur les rectangles, en utilisant les méthodes associées à la classe `Rectangle`. Dans une telle formulation, la relation (4) sera

définie par :

`not(intersects(r1, r2))`

où r1 et r2 sont des variables rectangles, et `intersects` est une méthode de la classe `Rectangle`.

Dans ce cas, le problème est défini par 2 variables, dont le domaine contient $N!/(N-4)!$ rectangles, avec $N = n^2 - n$, et une contrainte binaire, correspondant à la relation (4).

## 2.2. Comparaisons des deux formulations

Dans la première formulation, la relation (4) s'exprime difficilement. En effet, cette relation est énoncée naturellement en termes de rectangles, et sa traduction en termes de points oblige à un aplatissement difficile et coûteux de la relation. De plus, la définition des rectangles dans cette approche se fait via des contraintes (relation 3), ce qui interdit la réutilisation de classes préexistantes.
En revanche, la seconde formulation utilise des objets complexes dans les domaines des variables. Le langage induit par ces domaines (ici la méthode `intersects`) permet une expression naturelle des relations, via les méthodes de la classe `Rectangle`. Cette formulation est plus économique : la relation (4) étant définie par une contrainte binaire, au lieu d'une contrainte d'arité 8 dans le cas précédent. Ce point est capital, car l'arité des contraintes est un facteur déterminant pour l'efficacité de la résolution.
La Figure 3 illustre graphiquement les deux approches. Les '?' représentent les variables contraintes du problème. La première solution se trouve à gauche, avec ses rectangles partiellement instanciés, la seconde, à droite, avec ses variables contenant des objets rectangles.

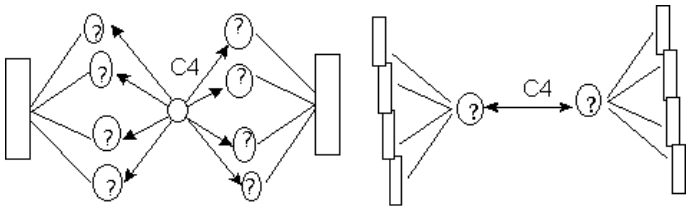


Figure 3: Illustration des deux approches pour la définition du problème de géométrie.

La difficulté, si l'on considère la seconde formulation, est qu'afin de définir les variables rectangles, il faut calculer un grand nombre d'objets : le cardinal du produit cartésien des variables points formant le rectangle,

i.e. le nombre de points à la puissance quatre ! Cette approche conduit donc à la création d'un CSP de taille rédhibitoire.

## 2.3. Notre approche

Nous présentons ici une approche qui permet d'utiliser la seconde formulation tout en évitant la création de trop nombreux objets. Ceci garantit la possibilité de réutiliser des classes préexistantes et de définir les contraintes de façon aisée par l'intermédiaire de méthodes. L'idée principale de cette démarche est qu'une partie des contraintes peut être remplacée avantageusement par des méthodes, et une partie des mécanismes de propagation par des instanciations d'objets. Dans le problème précédent, la relation (3) du problème d'origine, est prise en compte par la création d'instances de la classe `Rectangle`, au lieu d'être considérée comme une contrainte. Cette démarche se caractérise par la séparation en deux phases de la résolution du problème : 1) création et traitement d'un problème impliquant uniquement les objets simples et 2) construction d'un nouveau CSP, à partir du précédent, comprenant un nombre "raisonnable" d'objets complexes. C'est la résolution de ce second problème qui conclut. L'intérêt de cette séparation en deux problèmes est que la résolution du premier permet de réduire fortement le nombre d'objets structurés (ici les rectangles) du second problème, réduisant ainsi sa complexité.

Plus précisément, le premier problème est constitué de 4 variables $a, b, c$ et $d$ dont le domaine contient tous les $n.(n-1)$ points, et les contraintes sont : $x(a) = x(c); y(a) = y(b); x(b) = x(d); y(d) = y(c)$

L'énumération de toutes les solutions de ce CSP fournit la liste des rectangles admissibles par la suite, et qui seront traités par le deuxième problème. Le nombre de rectangles ainsi formés est de $n!/(n-4)!$ à comparer à $N!/(N-4)!$ avec $N = n^2 - n$.

Le second CSP est composé de 2 variables $r1$ et $r2$, dont le domaine est l'ensemble de rectangles précédent, et de la contrainte binaire `not(intersects(r1, r2))`. La résolution de ce second problème donne la liste des paires de rectangles satisfaisant à la définition de (P).
Ce schéma de résolution se généralise à des problèmes impliquant plus de deux niveaux de composition. Par exemple, le problème précédent peut posséder trois niveaux de composition si l'on considère les points comme des objets composites, les coordonnées devenant des objets atomiques. De façon générale, si un problème admet un nombre $n$ de niveaux de composition, on aura alors $n$ phases de résolution. Les avantages d'une

telle démarche de résolution en phases distinctes pour chaque niveau de composition sont les suivants :

1. La résolution d'un CSP en une seule phase oblige à fixer les heuristiques de résolution, dans notre cas, on peut adapter les heuristiques à chaque phase, voire changer d'algorithme de résolution. En particulier le deuxième problème étant plus fortement structuré, sa topologie sera vraisemblablement plus simple (moins de variables, et contraintes d'arité moindre).

2. Cette démarche permet la réutilisation de classes existantes, et la définition de contraintes utilisant directement leurs méthodes, au lieu d'avoir à définir des structures objets à l'aide de contraintes.

3. La quantité de contraintes et de variables nécessaires est moindre que lors d'une résolution en une phase. Sur ce problème particulier, la démarche présentée ci-dessus est bien plus efficace que la résolution classique n'utilisant pas de structures objets. La section suivante décrit un résultat similaire sur un problème plus complexe.

## 2.4. Complexité

La complexité algorithmique des deux approches peut être évaluée de manière simple. Dans la première, on a 8 variables (les points) de taille $n^2 - n$. Considérant que le traitement d'une contrainte consiste à calculer son produit cartésien, la complexité totale est donc de $(n^2 - n)^8$, soit de l'ordre de $n^{16}$. Dans la seconde approche, le premier problème comporte deux groupes de 4 variables de taille $n^2 - n$, reliées entre elles par une contrainte. Le calcul des objets rectangles du second problème coûte donc $2.(n^2 - n)^4$. Il en résulte un problème comportant deux variables, chacune de taille $n!/(n-4)!$, reliées par une contrainte binaire. Le coût global de résolution du second problème est donc de $(n!/(n-4)!)^2$, donc de l'ordre de $n^8$. Finalement, la complexité totale de la seconde approche est de l'ordre de $n^8$. Il y a donc un rapport quadratique entre les complexités des deux approches !

Ce résultat n'est valable ici que parce que l'ensemble des rectangles "plats" est beaucoup plus petit que l'ensemble des quadruplets de points possibles (il y a un rapport de racine carrée entre les deux). On peut tout de même évaluer le gain de la deuxième approche par rapport à la première dans le cas général. Considérons un ensemble $A$ d'objets atomiques, et un objet composite, constitué de $n$ objets atomiques ($n = 4$

pour le cas des rectangles). La taille de l'ensemble des n-uplets de A est alors $Card(A)^n$. L'ensemble des objets composites peut être vu comme l'image de $A^n$ par une fonction de composition, appelée $c$.

Considérant alors un problème impliquant deux objets composites (et donc $2.n$ objets atomiques), la complexité de la première approche est de l'ordre de $(Card(A)^n)^2$. En revanche, la complexité de la seconde approche est de l'ordre de $(Card(c(A^n)))^2$. Il est commode de considérer le rapport $r = Card(c(A^n))/Card(A)^n$. Ce rapport mesure la densité des objets composites dans l'espace de tous les n-uplets d'objets atomiques. Le rapport entre les deux complexités est alors de $r^2$. Il est clair que la deuxième approche est d'autant plus intéressante que r est petit devant 1, c'est à dire que les objets composites sont rares !

# 3. L'harmonisation musicale automatique

Dans cette partie, nous allons appliquer le schéma précédent au problème de l'harmonisation musicale automatique. Ce problème illustre parfaitement notre cadre de travail, puisque sa résolution dépend très fortement de la représentation choisie pour les structures d'objets musicaux.

## 3.1. Le problème

La composition et l'analyse musicales sont des domaines classiques de l'intelligence artificielle, et particulièrement dans la communauté de la programmation par objets, comme en témoignent les nombreux travaux dans ces domaines (e.g. Les systèmes MODE [26], Formes [8], Kyma [30], etc.).

Nous considérons ici le problème de la génération de pièces polyphoniques à 4 voix à partir d'un matériau musical incomplet (généralement seule la mélodie, ou la basse, est donnée), respectant les règles de l'harmonie tonale, telles que l'on peut les trouver dans un traité d'harmonie [4]. Des exemples typiques de règles sont "deux accords consécutifs ne peuvent comporter, entre les mêmes voix deux quintes ou deux octaves", ou bien " la note sensible doit monter obligatoirement à la tonique", ou encore "deux accords consécutifs doivent être de degré distinct".

La formulation de cet exercice comme un problème de satisfaction de contraintes est une approche déjà "classique" (Cf. 3.2) dans laquelle

les contraintes représentent les règles musicales et les variables correspondent aux structures musicales (notes, intervalles, accords). Cette possibilité vient de ce que les règles harmoniques sont généralement exprimées sous la forme de combinaisons interdites, ce qui convient particulièrement bien au formalisme des contraintes (Cf. [21]).

Ce problème est par ailleurs difficile du fait de sa grande combinatoire : si l'on considère que chaque note à trouver peut prendre sa valeur dans un domaine de 14 notes (la tessiture de la voix considérée), et que pour une mélodie de $n$ notes on doit calculer $3.n$ notes, la taille de l'espace de recherche est de l'ordre de $14^{3.n}$. Une mélodie d'exercice standard comporte typiquement 15 notes, conduisant à $10^{52}$ combinaisons possibles.

En outre, ce problème implique des structures complexes (accords, gammes etc.). De plus, les règles harmoniques, et donc les contraintes, définissent un ensemble hétérogène de relations : relations horizontales entre notes successives, relations verticales entre notes simultanées, relations complexes entre accords successifs, etc. La Figure 4 donne un aperçu, sur une partition complète, de la topologie des différentes règles harmoniques.

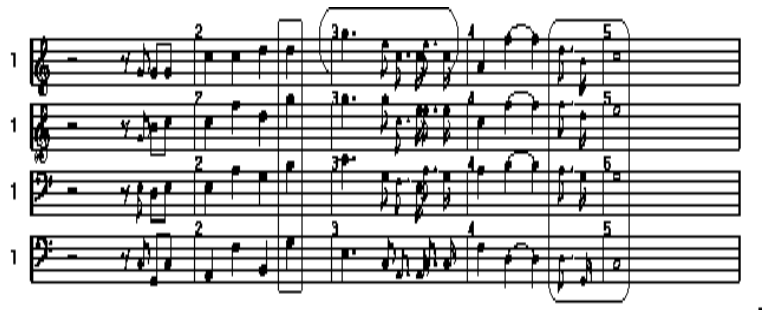

Figure 4: Illustration des différents types de règles harmoniques.

Sur la Figure 4 on voit que certaines règles sont horizontales (mélodiques, comme "la sensible doit monter à la tonique"), alors que d'autres sont verticales (harmoniques, comme "pas d'intervalle de triton, sauf pour les accords de dominante"). De plus, certaines portent sur des structures simples, les notes, alors que d'autres portent sur des structures complexes, les accords ("deux accords consécutifs doivent être de degré différents"). La grande difficulté de ce problème, à la fois dans sa représentation et sa résolution, vient de l'interaction entre ces contraintes de natures différentes, et qui portent sur des structures "orthogonales"

comme les mélodies et les accords. Cette difficulté est naturelle puisque l'objet d'un exercice d'harmonie est justement de construire une pièce dans laquelle cohabitent des mouvements mélodiques (horizontaux) imposés, avec des structures harmoniques (verticales).

## 3.2. L'harmonisation musicale et la satisfaction de contraintes

Comme nous venons de le dire, la représentation des règles musicales par des contraintes est naturelle, ce qui a conduit à la réalisation de nombreux systèmes d'harmonisation automatique. La première tentative est celle de Ebcioglu [10], qui a conçu un langage de programmation logique avec contraintes, BSL, utilisé pour la génération de chorals dans le style de Jean-Sébastien Bach. Ce système, outre la possibilité de génération de chorals à partir d'une mélodie donnée, peut aussi construire des pièces complètement nouvelles. L'architecture de BSL est intéressante, mais elle s'appuie sur des mécanismes sensiblement différents de ceux que nous utilisons. En effet, les contraintes sont utilisées de manière passive par un algorithme de type 'retour-arrière intelligent', qui est moins efficace, bien que très sophistiqué, que les algorithmes basés sur la propagation de contraintes (forward-checking, real-full look-ahead [22], [27], par exemple).

Plus récemment, [31] ont présenté un système résolvant un problème identique au nôtre avec un langage de CLP. Leurs résultats ne sont pas encourageants (5 minutes et 70 Mo de mémoire vive, pour une mélodie de 11 notes), et interdisent un usage effectif du système.

Ovans [23] fut le premier à résoudre ce problème avec des mécanismes basés sur l'arc-cohérence ([21]), qui ont pour effet d'améliorer sensiblement les performances. Cependant, on peut déplorer la difficulté de représentation des contraintes musicales et la pauvreté des structures musicales employées, le système de Ovans étant seulement apte à manipuler des nombres entiers. À titre d'exemple, la contrainte qui représente la règle interdisant les quintes et octaves parallèles est reproduite ci-dessous :

$$parallelfifth(c_i, m_i, c_{i+1}, m_{i+1})$$

$$\Leftrightarrow$$

$$\neg(perfect(c_{i+1}, m_{i+1}) \vee (c_i - c_{i+1}).(m_i - m_{i+1}) \leq 0$$

avec

$$perfect(c_i, m_i) \leftrightarrow \mid c_i - m_i \mid \in \{0, 7, 12, 19\}$$

Dans ces énoncés, les $m_i$ désignent les notes de la mélodie, et les $c_i$ désignent celles du contrepoint. La valeur 7 désigne la quinte, 12 l'octave, etc.

Le système de Ballesta [2], écrit en Pecos, peut être considéré comme une ultime évolution des réalisations précédentes. Comme Ovans, Ballesta utilise des algorithmes efficaces basés sur une utilisation active des contraintes, mais emploie de plus une représentation très fine des structures musicales. Il s'agit cependant ici d'un système de contraintes "pur" dans lequel la notion d'objet recouvre en fait un ensemble d'attributs liés entre eux par des contraintes. Cette représentation est extrêmement riche et dynamique puisqu'on peut employer des objets partiellement instanciés. Par exemple, une instance de la classe `Intervalle` est définie par 12 attributs comme son nom, son type, ses notes extrêmes etc., et des contraintes assurent que ces attributs forment un intervalle. Dans cette représentation, il est possible de demander à un intervalle de donner la liste des notes qui en forment une tierce avec une note donnée (comparer avec le système MusES, présenté au 3.4). Mais cette richesse se paye, en revanche, lors de la représentation d'un problème concret, puisque chaque entité musicale est à elle seule un CSP. Le système crée, pour une mélodie donnée de $n$ notes, un problème comprenant $126.n - 28$ variables contraintes !

## 3.3. Une nouvelle approche

On peut tirer deux enseignements de l'analyse des systèmes réalisés jusqu'à présent :

1) Ils sont essentiellement basés sur les contraintes, et n'exploitent donc pas de représentation riche des structures musicales. Seul le système de Ballesta dispose de représentations structurées des objets musicaux, mais elle sont elles-mêmes réalisées par des contraintes.

2) Les contraintes sont traitées de façon uniforme, et dans une unique phase de résolution. Ceci reflète mal la réalité : un musicien raisonne selon différents niveaux d'abstraction mettant en jeu des structures de complexités différentes (d'abord les notes, puis le raisonnement se concentre sur les accords). La prise en compte de cette séparation naturelle apporte un fort gain en efficacité, comme nous allons le montrer.

Ces remarques nous conduisent à aborder le problème en partant des objets, et en leur adjoignant des contraintes de façon adaptée, plutôt que de partir des contraintes, et de construire des structures musicales

idoines. De plus, les structures harmoniques se prêtent bien à une représentation par classes, au sens de la programmation par objets. Nous sommes donc partis d'une librairie de classes musicales fournissant des représentations des notes, des intervalles, des accords et des gammes : MusES [24]. Les contraintes ont ensuite été définies, dans le système BackTalk, entre ces objets.

## 3.4. La librairie MusES

MusES est une librairie de classes représentant les concepts usuels de la musique tonale : notes, enharmonies, gammes, accords, analyses, tonalités, etc. MusES contient environ 100 classes Smalltalk et 1500 méthodes, et utilise uniquement les mécanismes standards de la programmation par objets : instanciation, héritage, polymorphisme, etc. Comparée à une représentation par contraintes, cette approche est moins générale. En effet, au lieu de définir des relations non directionnelles entre objets, MusES propose une batterie de mécanismes "essentiels", implémentés de façon simple et efficace. La classe des intervalles fournit une bonne illustration de cette approche, à comparer avec celle de Ballesta (cf. 3.2). L'idée est que trois opérations seulement sont utilisées en pratique sur les intervalles : 1) calculer l'intervalle entre deux notes données, 2) calculer l'origine d'un intervalle en connaissant son extrémité et 3) calculer l'extrémité connaissant l'origine. MusES propose ainsi une méthode pour chacune de ces trois opérations.

## 3.5. Un système efficace pour l'harmonisation automatique

Il existe un forte analogie entre le problème d'harmonisation et celui des rectangles présenté au 2 : les notes correspondent aux points, et les accords aux rectangles. Cette analogie vaut également pour les contraintes : il y a des contraintes entre notes ainsi que des contraintes entre accords.

Cette analogie nous conduit à appliquer la démarche présentée en 2.3 à la résolution du problème d'harmonisation, qui se décompose alors comme suit :

1) Création puis résolution du CSP-1, ne contenant que les variables notes et les contraintes sur les notes.

2) Instanciation des accords possibles à partir du résultat précédent, i.e. les notes restant dans les domaines des variables du CSP-1, et con-

| | 11 notes | 12 notes | 16 notes |
|---|---|---|---|
| **Tsang & Aitken** | 1 min. 70 Mo * | ? | ? |
| **Ballesta (Pecos)** | ? | 3 min. | 4 min. |
| **BackTalk + MusES** | 1 sec. | 3 sec. | 4 sec. |

Table 1: Résultats sur station SUN Sparc 10. * est un temps extrapolé à partir d'un temps publié de 5 mn sur Sparc1.

struction du CSP-2, contenant les accords et les contraintes sur les accords. Résolution de ce CSP.

Cette approche conduit à la même amélioration de complexité que celle vue au paragraphe 2.4. Ici, le nombre d'accords créés est très inférieur au nombre de quadruplets de notes possibles : il existe environ $14^4$ quadruplets de notes possibles, à comparer aux "seulement" quelques centaines d'accords licites (ce chiffre varie en fonction de la note du soprano et de la tonalité).

Ce schéma de résolution produit un gain en efficacité remarquable (Cf. tableau 1) par rapport à une résolution classique en une seule phase.

Par ailleurs, cette approche nous a permis de réutiliser telles quelles les classes de MusES, économisant ainsi un profond travail de représentation des structures harmoniques, et nous permettant des définitions de contraintes très naturelles et intelligibles tout en étant plus efficaces. Par exemple, la contrainte interdisant à deux accords consécutifs d'avoir des quintes ou des octaves parallèles sera simplement posée de la façon suivante (à comparer au même exemple chez Ovans en 3.2) :

```
BTConstraint
      on: accordCourant
      and: accordSuivant
      block: [:c1 :c2 |
         c1 hasNoParallelPerfectIntervalWith: c2].
```

La Figure 5 montre les différents systèmes mentionnés dans ce papier, disposés suivant l'importance relative qu'ils donnent aux contraintes et aux objets. MusES est une librairie de représentation de connaissances purement objets, tandis que le système de Tsang et Aitken n'utilise pas d'objets. Entre les deux se trouvent à la fois notre système, combinant des contraintes avec des objets classiques, et le système de Ballesta dont

les objets sont définis par des contraintes. Le meilleur compromis est obtenu avec un système intermédiaire, qui combine les avantages des deux paradigmes.

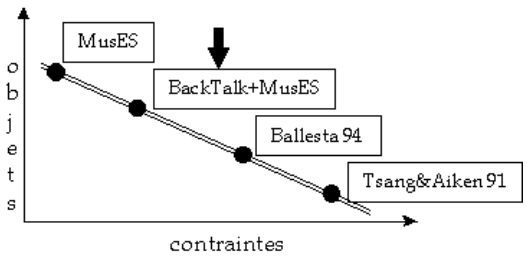


Figure 5: Différents systèmes d'harmonisation automatique classés en fonction de leur utilisation de contraintes et d'objets.

# 4. Conclusion

L'intégration des mécanismes de satisfaction de contraintes dans les langages à objets pose d'importantes difficultés de conception. Nous avons mis en évidence, à l'aide d'un exemple simple de géométrique plane, un problème crucial qui surgit naturellement lors de la formulation de CSPs impliquant des structures objets complexes. Nous avons présenté et comparé deux solutions opposées pour résoudre cette difficulté, et proposé une démarche de résolution permettant de combiner leurs avantages. Cette solution consiste à séparer le problème en plusieurs phases distinctes, correspondant aux divers niveaux de composition du problème, et à résoudre chaque problème intermédiaire pour réduire le domaine des variables du problème suivant.

Cette solution permet d'obtenir un gain en efficacité, parce que certaines contraintes sont remplacées par des méthodes associées aux classes d'objets mises en jeu. Elle permet aussi de réutiliser des librairies de classes non conçues a priori pour résoudre le problème. Nous avons montré que la résolution du problème complexe de l'harmonisation musicale automatique par cette approche permet de diviser par au moins un facteur 10 le temps d'exécution, et de simplifier considérablement l'écriture des contraintes, validant ainsi notre approche.

Nous étudions la généralisation de notre approche à d'autres problèmes de contraintes et objets, en particulier les problèmes de configura-

tions.

## Bibliographie

[1] Avesani, P. Perini, A. Ricci, F. (1990) COOL : An Object System with Constraints. Proceedings of TOOLS'2, Angkor, Paris, pp. 221-228.

[2] Ballesta, P. (1994) Contraintes et objets : clefs de voûte d'un outil d'aide à la composition ? Ph.D. Thesis, INRIA, Sophia Antipolis, November 1994.

[3] Berlandier, P. (1992) Etude de mécanismes d'interprétation de contraintes et de leur intégration dans un système à base de connaissances. Ph.D. Thesis, INRIA, 1992.

[4] Bitsch, M. (1957) Précis d'Harmonie tonale. Ed. Alphonse Leduc.

[5] Borning, Alan, H. (1981) The programming language aspects of ThingLab, a constraint oriented simulation laboratory. ACM transaction on Programming Languages and Systems, 3 (4) pp. 353-387.

[6] Borning, A & Freeman-Benson, B. (1995) The OTI Constraint Solver : A Constraint Library for Constructing Interactive Graphical User Interfaces. CP'95, Springer-Verlag, Lecture Notes in Computer Science n. 976. U. Montanari & F. Rosi Eds, pp. 624-628.

[7] Caseau, Y. (1994) Constraint Satisfaction with an Object-Oriented Knowledge Representation Language. Journal of Applied Artificial Intelligence, 4, pp. 157-184.

[8] Cointe, P. Rodet, X. (1991) Formes : Composition and Scheduling of Process. In The Well-Tempered Object : Musical Applications of Object-Oriented Software Technology , S. T. Pope, ed. MIT Press.

[9] Colmerauer, A. (1990) An introduction to Prolog-III. Communications of the ACM, 33 (7) : 69.

[10] Ebcioglu, K. (1992) An Expert System for Harmonizing Chorales in the Style of J.-S. Bach, In M. Balaban, K. Ebcioglu & O. Laske (Ed.), Understanding Music with AI : Perspectives on Music Cognition, The AAAI Press, California.

[11] Freeman-Benson, B. (1990) Kaleidoscope : mixing objects, Constraints, and Imperative Programming. Proceedings of ECOOP/OOPSLA 90, Ottawa, pp. 77-88.

[12] Freeman-Benson, B. Borning, A. (1992) Integrating constraints with an object-oriented Language. Proceedings of ECOOP '92, Utrecht, Springer-Verlag Lecture Notes in Computer Science, vol. 615, pp. 268-286.

[13] Gensel, J. (1995). Integrating Constraints in an Object-Based Knowledge Representation System. In Lecture Notes in Computer Science n. 923, Springer-Verlag, pp. 67-83.

[14] Jaffar, J. Lassez, J.-L. (1987). Constraint logic programming. 14th POPL, Munich, 1987.

[15] Karsenty, S. Weikart, C. (1994). Rollit : An Application Builder. Proceedings de TOOLS'13, Prentice-Hall, pp. 15-26.

[16] Kökény, T. (1994). Yet another object-oriented constraint resolution system (YAFCRS) : an open architecture approach, Proceedings of TOOLS USA 94, Prentice-Hall, Santa Barbara, pp. 103-114.

[17] Lalonde, W. Van Gulik, M. (1988). Building a backtracking facility for Smalltalk without kernel support. Proceedings of OOPSLA '88, San Diego (Ca), pp. 105-123.

[18] Laurière, J.L. (1978). A language and a program for stating and solving combinatorial problems. Artificial Intelligence, Vol. 10, pp. 29-127.

[19] Liotard, J. (1994). Contraintes, objets et application : la CAO. Proceedings de LMO'94, Grenoble, pp. 181-192.

[20] Lopez, G. Freeman-Benson, B. Borning, A. (1994). Constraints and Object Identity. Proceedings of ECOOP '94, Bologna (Italy), Springer-Verlag Lecture Notes in Computer Science, vol. 821, pp. 260-279.

[21] Mackworth, A. (1977). Consistency on networks of relations. Artificial Intelligence, (8) pp. 99-118, 1877.

[22] Nadel, B. (1988) Tree search and arc-consistency in constraint satisfaction algorithms. Search in Artificial Intelligence, Springer-Verlag, pp. 287-340.

[23] Ovans, R. (1992) An Interactive Constraint-Based Expert Assistant for Music Composition. Proc. of the Ninth Canadian Conference on Artificial Intelligence, University of British Columbia, Vancouver, 1992.

[24] Pachet, F. (1994) The MusES system : an environment for experimenting with knowledge representation techniques in tonal harmony. First Brazilian Symposium on Computer Music - SBC&M '94, August 3-4, Caxambu, Minas Gerais (Brazil), pp. 195-201.

[25] Pachet, F. Roy, (1995) P. Mixing constraints and objects : a case study in automatic harmonization. Proceedings of TOOLS Europe '95, Versailles, Prentice-Hall, pp. 119-126.

[26] Pope, S. (1991) Introduction to MODE : The Musical Object Development Environment. In The Well-Tempered Object : Musical Applications of Object-Oriented Software Technology , S. T. Pope, ed. MIT Press.

[27] Prosser, P. (1993) Hybrid algorithms for the constraint satisfaction problem. Computational intelligence, Vol. 9, pp. 268-299.

[28] Puget, J.-F. (1994) Programmation logique sous contraintes en C++. Proceedings of "Langages et modèles à objets" LMO'94, Grenoble (France), pp. 193-202.

[29] Roy, P. Pachet, F. (1996) Reifying Constraint Satisfaction in Smalltalk. Journal of Object-Oriented Programming, à paraître.

[30] Scaletti, C. Johnson, R. E. (1988) An interactive environment for object-oriented music composition and sound synthesis. Proceedings of OOPSLA '88, pp. 222-233, San Diego.

[31] Tsang, Chi Ping & Aitken, M. (1991) Harmonizing music as a discipline of constraint logic programming. Proceedings of ICMC '91, Montréal, pp. 61-64.

[32] Van Hentenryck, P. (1989) Constraint satisfaction in Logic programming. MIT Press, Cambridge, MA, USA.