Formulating Constraint Satisfaction Problems on Part-Whole Relations: The Case of Automatic Musical Harmonization

François Pachet[1], Pierre Roy[2]
[1]SONY CSL-Paris, 6 rue Amyot, 75005 Paris, France, pachet@csl.sony.fr,
[2]LIP6, Université Paris 6, E-mail : roy@poleia.lip6.fr

Abstract:
We address the issue of formulating constraint problems in highly structured domains, and focus on the exploitation of part-whole relations. We describe and compare two approaches in problem formulation in this context ; one using a flattened representation of domains, and one using a naturally occurring part-whole relation. We show that the second approach may be much more efficient than the flattened representation, thanks to a judicious partitioning of the constraint problem in several phases. We give a theoretical measure of the respective complexities of the two approaches, and illustrate our result on a difficult and naturally structured musical problem: automatic harmonization of melodies. The resulting system outperforms previous attempts on the same problem. Finally, we propose other applications of our approach for generating automatically musical programs.

1. Formulation of Constraint Satisfaction Problems

Constraint satisfaction techniques have been increasingly used for solving difficult combinatorial problems recently, thanks to the availability of efficient algorithms and data structures. However, most of the problems solved by these techniques are still problems traditionally belonging to the field of operational research. These problems are usually formulated using a rather formal representation, such as graph theory, and constraint problems are typically designed with integer or real variables.
We are interested in the application of constraint satisfaction techniques in other fields, where domains are already naturally structured. This is typically the case for instance of multimedia, where there 1) is a growing pool of new problems which call for efficient solving techniques, and 2) the underlying ontologies are not easily reduced to mathematical entities such as integer or real numbers.

In this context, we are interested in the formulation of constraint problems in spaces where either domains are smaller or more knowledge is available. This is possible only for problems involving structured domains. Usually problems coming from the field of operational research are already formalized to a point where efficient specialized algorithms can be used, on a “flattened” representation of the problem. Typically this is the case for scheduling problems, where objets to be scheduled are faithfully represented as integers, without losing any relevant information.
In naturally structured domains such as multimedia, we claim that there are lots of problems which 1) can be efficiently modeled using constraints, and 2) handle highly structured domains. This is typically the case of the automatic harmonization problem, on which we will concentrate in this paper. However, this paper is not specifically about harmonization, but about formulation of constraint problems in naturally structured domains.

In section XX we will describe the problem, and review preceding approaches for solving it. In section XX we will describe our approach, based on a formulation of the problem in a different space. In section XXX we will evaluate the theoretical complexity of our approach. We will conclude on the generality of our proposal, and other applications in progress.

2. The Automatic Harmonization Problem

Solving a harmony exercise consists in finding a harmonization of a given melody, such as the melody shown by Figure 1, or, more generally, any *incomplete* musical material. This harmonization must satisfy the rules of harmony (and counterpoint, if rhythm is taken into account). These rules are consensual, and can be found in any decent treatise on harmony (e.g. [Bitsch 1957]). A standard exercise is to harmonize a given melody into a four-voices harmonization (see Figure 2 for a solution of the melody of Figure 1).

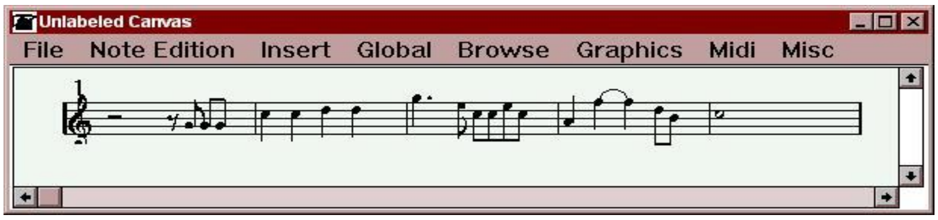

Figure 1 An initial melody to harmonize (the beginning of the French national anthem, 18 notes).

The main interest of this musical problem for CSP is that the rules of harmony and counterpoint fit nicely in the formalism of finite domain CSP. There are various types of such constraints: 1) horizontal constraints on successive *notes* of a melody (e.g. “two successive notes should make a consonant interval”), 2) vertical constraints on the *notes* making up a *chord* (e.g. “no interval of augmented fourth, except on the fifth degree” or “voices never cross”) and 3) constraints on *sequences of chords* (e.g. “two successive chords should have different degrees”).

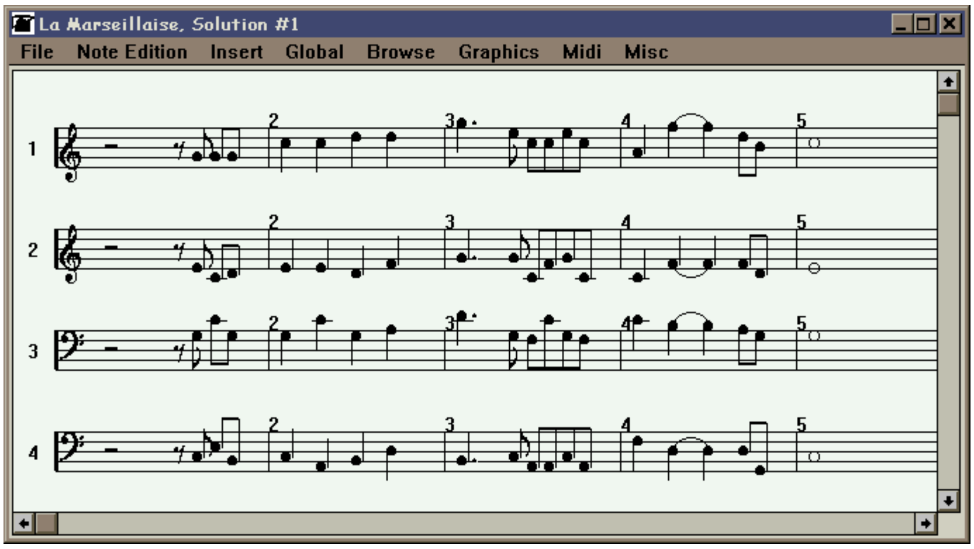
La Marseillaise, Solution #1
File Note Edition Insert Global Browse Graphics Midi Misc
1
2
3
4
5

Figure 2. A solution proposed by BACKTALK *from the initial melody of Figure 1.*

The problem naturally involves the use of constraints, because of the way the rules of harmony are stated in the textbooks. It is not surprising that several studies have proposed various solutions to solve the automatic harmonization problem (AHP) using constraints. We will review them in the following section, and outline their main characteristics.
Here, “Satisfactory” means findings solutions instantaneously !!!!!

3. Previous Approaches to the AHP

The first system historically, who solved the AHP way probably Ebcioglu’s Choral system [Ebcioglu 1992]. Choral‘s objective was to generate full fledged chorales in the style of J.S. Bach from scratch. To achieve this aim, Ebcioglu devised an ad hoc constraint logic programming language (BSL), based on production rules together with a backjumping technique. Ebcioglu’s system included lots of different and interacting modules, including melody generation, harmonization, and analysis, as well as modules to consider stylistic constraints. It is therefore difficult to asses the quality of the harmonization module only as such. From a technical point of view, constraints were used mostly *passively*, to reject solutions produced by production rules.

Steels approach (Steels 86) was aimed at building a system that could learn the rules of harmony itself, using machine learning techniques. Although not primarily designed for automatic, efficient harmonization, the system featured an interesting capacity to learn and use musical heuristics to speed up the search, and an ability to combines brute exploration with heuristic search, a important concern of CSP in the large.
Ovans (92a, 92b) was one of the first to advocate the use of arc-consistency techniques, now widespread in constraint satisfaction. He showed experimentally that techniques based on arc consistency were more adapted and efficient than techniques based on earlier techniques, such as backjumping. However, Ovans used a flat

representation of the harmonization problem (2 voice only), and his results, although better than his predecessor's were still unsatisfactory (in the sense of the preceding section).

Tsang & Aitken (91) proposed a solution of the four-voice harmonization problem using constraint logic programming techniques [Jaffar and Lassez 1987]. Although the proposed system was operational, the authors concluded on the impracticability of the approach, for efficiency reasons : it took more than 1 minutes and 70 Megabytes of memory to solve the AHP on an 11-note melody. They employed also a flat representation of the musical domain : constraints were stated on representations of "notes".

More recently, Henz et al. (96) addressed the harmonization problem from a different viewpoint, emphasizing the need for developing a plan, prior to the resolution. This plan embodies knowledge about the desired harmonization, such as modulations and chord degrees. The harmonization problem is therefore much simpler, and solutions are found in reasonable time (about 1 second for non optimal solutions on a 133mhz PC). The resulting system is interesting from a musical point of view because it allows users to effectively test harmonization plan interactively. However, from the constraint satisfaction point of view there is no contribution since basic techniques (the Oz system XXXX) are used to solve a much simpler problem.

The system proposed by Ballesta [Ballesta 1994] is probably the most complete attempt at solving the four-vioce harmonization system using recent constraint techniques (a Lisp version of the Ilog Solver system). The approach taken by Ballest, howevern is the same than his predecessors: the musical domain is represented as objects, but the constraints are all stated on the flattened representation of these objects. For instance, in Ballesta's system, 12 attributes are defined to represent one interval, such as the *name* of the interval (e.g. diminished fourth), its *type* (e.g. fourth), its two extremities, represented as instances of class `Note`. Nine constraints are then introduced to state the relations that hold between the various attributes of class Interval. For instance, a constraint links the name of the interval to the various attributes of its extremities (the octave and name of the note). As a result, constraints have to be defined using a low-level language (i.e. arithmetic), thus requiring a translation of harmonic and melodic properties, given in harmony treatises, in terms of numbers. The constraint representing the rule expressing that relations of parallel fifth are forbidden between two successive chords is given below as it is expressed in Ovan's system. XXXPeut on prendre un exemple de Ballesta plutot ?

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

where

$$\texttt{perfect}(c_i,m_i) \Leftrightarrow |c_i-m_i| \in \{7,19\}$$

Figure 3. The constraint corresponding to the parallel fifth rule, expressed in Ovan's system, based on CLP.

It is clear that this approach leads to stating a huge amount of constraints and constrained variables. For instance, in Ballesta's system, one note instance is represented by 6 constrained variables, one interval by 9 constrained variables. To solve the AHP on a *N*-note melody, Ballesta uses $126 \times N - 28$ constrained variables. The resulting system is expectedly slow, and shows clearly - through the construction of a full system - the limits of the "flattened" approach.

4. Exploiting Natural Part-Whole Hierarchies in Formulating CSPs

The poor performance of the preceding systems, led us to experiment a radically different approach. The drawbacks of these systems can be summed up as follows: first, there are too many constraints. The approaches proposed so far do not structure the representation of the domain objects (notes, intervals, chords). When such a structure is proposed (as in Ballesta's system) objects are treated as passive clusters of constrained variables. Second, the constraints are treated uniformly, at the same level. This does not reflects the reality: a musician reasons at various levels of abstraction, working first at the note level, and then on the chords. The most important harmonic decisions are actually made at the chord level. This separation could be taken into account to reduce the complexity.
Our basic idea is to exploit a natural structuring of objects in this problem. Here, the main natural structure, on top of notes, are chords. Chords can be seen as groups of simultaneous notes having certain properties. The main

interest of chords is that they form meaningful entities on which knowledge is usually formulated. For instance, the parallel fifth constraint typically holds between two chord objects, and not between 8 notes !
In order to formulate precisely this idea, identify the technical problems and propose a solution, we will concentrate on a somewhat simpler problem in the field of elementary geometry in the next section. We will then apply the solution to the harmonization problem, and describe the resulting system.

4.1 Definition of a CSP

Let us first define a CSP using the extensional formulation.

Definition 1 Extensional definition of a CSP. *Let E be a finite set. A* CSP *P on E consists of the following elements:*
- A set $V=\{v_1, \ldots, v_n\}$; the v_i are the variables of P
- A mapping $\mathcal{D}:V \rightarrow \mathcal{P}(E)$; $\mathcal{D}(v_i)$ is the domain of v_i. *We call D the Cartesian product of all the domains: $D=\mathcal{D}(v_1)\times\ldots\times\mathcal{D}(v_n)$*
- A finite set C. Elements of C are the constraints
- A mapping $r: C \rightarrow \mathcal{P}(D)$; $\forall c \in C$; r(c) is the set of tuples satisfying c, and is called the extension *of c.*
OU :
a mapping $I: C \rightarrow \{1..\mathfrak{n}\}$; $\forall c \in C$; I(c) is the set of the indices of the constrained variables involved in C
XXXXa mapping r: C→ ???; $\forall c \in C$; ; r(c) is the set of tuples satisfying c, and is called the extension *of c.*

In this definition, it is implicitly considered that every constraint involves all the variables of *P*. In the rest of this paper, the expression "the variables of constraint *c*" denotes the variables actually involved in the constraint.

Definitions 2 Instantiation and solution. *An* instantiation *of the variables is any $s \in D$. A* solution s *of P is an instantiation such that $s \in r(c)$ for every $c \in C$. (i.e. s satisfies all the constraints) The solution set of P will be denoted by S(P) or simply S.*

4.2 Two main designs for CSP and structured domains

We will now illustrate our problem by showing two different formulations of a simple problem in the domain of planar geometry. The example is the following:

Problem Statement (P)
Find all pairs of non trivial quadrilaterals satisfying the following set of constraints (C) :

C1 - All vertices have integer coordinates in {1 .. n}.
C2- For all vertices, the x and y coordinate are different.
C3 - All quadrilaterals are straight rectangles.
C4 - The two rectangles do not intersect.

Figure 1 shows a solution of (P) when n = 10.

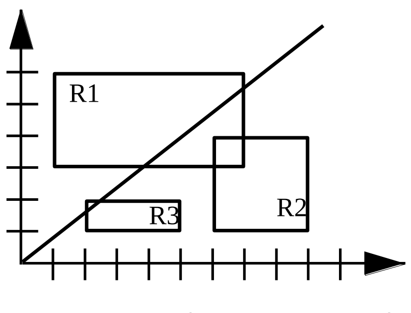

Figure 1. A solution of problem (P) is a pair of rectangles satisfying the set of constraints (C). The pairs of rectangles {R1, R3} and {R2, R3} are possible solutions; pair {R1, R2} is not. In a space with coordinate ranging from 1 to 6, there are 90 solutions.

4.2.1 Two formulations

The main task in formulating problem P is to identify the variables to be constrained. The first formulation consists in “flattening out” the domains, i.e. specifying the problem only with point variables, and considering points as atomic entities[1]. In this representation the constraints are stated entirely in terms of point variables. For instance, constraint C4 could be stated as follows:

```
x(b) < x(a')       "R1 on the left of R2"
or x(a) > x(b')    "R1 on the right of R2"
or y(c) > y(a')    "R1 above R2"
or y(a) < y(c')    "R1 below R2"
```

where a, b, c, d (resp. a', b', c', d') are the coordinates of R1 (resp. R2), and x and y are the functions yielding the x and y coordinate of the points (Cf. figure 2).

In this solution there are :

- 8 variables, representing the 8 vertices, each one with a domain of size (n^2 - n).
- 9 constraints (C1 and C2 are represented as domains, 8 binary constraints for expressing C3, one 8-ary constraint for C4).

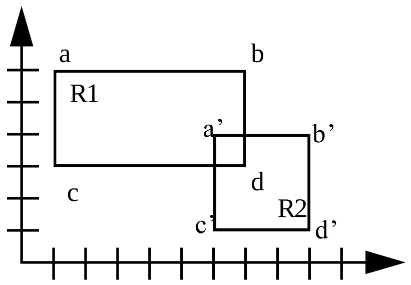

Figure 2. Stating constraint C4 in terms of points.

The second formulation is based on the exploitation of natural structures : the rectangles themselves. Since the problem statement contains constraints involving rectangles (C4) and that the domain is defined in terms of `Point` objects, a natural solution consists in specifying a problem with *both* "rectangle variables" (i.e. variables whose domain is collection of rectangle objects) and point variables. The main interest of this solution is to allow the statement of constraints in a more natural manner. For instance, constraint C4 could then be stated as :

```
not (intersects (r1, r2)),
```

where r1 and r2 are rectangle objects.

In this solution, notwithstanding the other constraints of problem P, there would be :

- 2 variables, representing the 2 rectangles, each one with a domain of size N! / (N-4)!
  where N = n^2-n (the number of possible points).
- 1 binary constraint, for C4.

However, this formulation raises two problems : how to handle the remaining constraints (C1, C2? C3), and how to build rectangle objects to create the domains for these variables ?

In the first solution, constraints are difficult to state, because they involve only "lower-level" objects. In our previous example, imagine a constraint involving three rectangles! Moreover, classes defining the domain objects (here class `Point` and class `Rectangle`) may not be simply reused for the problem solving. In our

[1] We will see later that our architecture can accomodate the case when only integer are considered atomic, and points considered as aggregates.

example, class `Rectangle` is "by-passed" by the definition of constraints (more precisely, constraint C4 bypasses method `intersects`).

In the second solution, constraints are stated using higher-level objects (here class `Rectangle`). These higher-level objects are used as such, (method `intersects` in class `Rectangle`), and play a central role in the formulation. Also, the constraints involve less variables (recall that the arity of constraints is a crucial parameter for performance). In our example, C4 has 8 variables in the first solution, and only 2 in the second one.
The problem is that rectangle variables have no domain a priori. Therefore a lot of objects have to be created: in our case it is roughly equal to the Cartesian product of domain sizes of the variables making up a rectangle. Although the collection of all possible rectangles in a finite 2-dimension space is, of course, finite, it is practically unreasonable to build this collection prior to the resolution.

These two approaches are graphically represented in Figure 3. The first one corresponds to a problem with partially instanciated objects, i.e. objects whose attributes are constrained variables. In the second one, constraints are stated between fully instanciated objects.

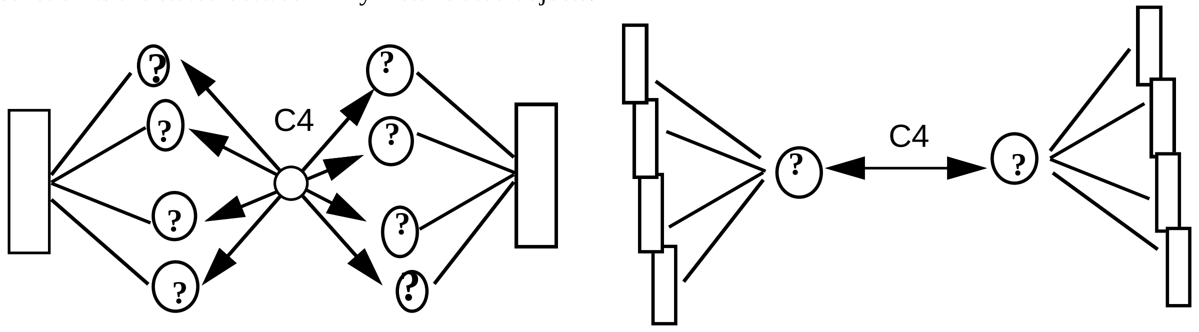

Figure 3. The two design approaches in object + finite CSP problem solving. Solution 1 is on the left, solution 2 on the right.

4.3 Introducing a Part-Whole Relation in the Formulation

We will now introduce the part-whole relation between domain values in the definition of the CSP. The idea is that a part-whole relation yields a natural partition of the set of variables and of the set of constraints.
We will consider here only "one-level" part-whole relationships.
The generalization to arbitrary levels is trivial.
The relation between domain values is transposed to variables.

Definition 1 Part-Wholed CSP. *Let P a* CSP.
A part-whole relation R is defined as follows:

- $V = V1 \cup V2$
- $R : V1 \times V2 \rightarrow \{0, 1\}$
- *v1 R v2 means that v2 is a "part of" v1*

This relation between variables induces a partition of the constraints in three sets, according to the nature of their variables. This partition is defined as follows:

$$C = C_1 \cup C_2 \cup C_{mixed} \text{ with } C_1 \cap C_2 = \varnothing \text{ and } C_1 \cap C_{mixed} = \varnothing \text{ and } C_2 \cap C_{mixed} = \varnothing$$

where the subset C_1 (resp. C_2), consists of constraints involving only variables of V1 (resp. V2), and where C_{mixed}, contains variables of both V1 and V2.

Our proposal consists in decomposing the resolution of P into the successive resolution of sub-problems holding only on subparts of C.

Based on the preceding discussion, we propose a scheme that combines the advantages of the second solution (declarativity, possibility of reusing existing classes, constraints with lesser arity), while avoiding the systematic creation of all higher-level objects.

The strategy we came up with is to separate the resolution into two different phases: in a first time, we consider only atomic objects of the problem and the constraints involving them; we then use domain reduction to remove from this initial problem values that are not consistent; finally, we compute all the structures that can be made up with the remaining atomic values. The remarkable thing is that the number of possible higher-level objects has become reasonable.

Our scheme is based on the idea that the number of higher-level objects can be reduced by solving a preliminary CSP with only constraints over lower-level objects. The solutions of this preliminary CSP constitute the domains of the variables representing the higher-level objects.

The strategy can be formalized as follows:
First phase
We first create a CSP P1, whose variable set is V1, and whose constraint set is C_1.
We apply a consistency algorithm to P1, in order to reduce the size of the domains of its variables. The more reduced the domains are, the better.

Intermediate phase
The intermediate phase consists in reducing the domain of every v2 in V2. To do so, we just restrict the domain of v2 with respect to the new, reduced, domain of each v1 that satisfies v1Rv2.

Second phase
We then create a CSP P2, whose variable set is $V=V1\cup V2$, and whose constraint set is $C_2\cup C_{mixed}$. The variables in V2 have has their domain reduced in the intermediate phase, and are not huge anymore.

In our rectangle example, the first CSP is the following :

CSP P1
- variables: `a, b, c, d` whose domains are the list of all possible points (size n(n-1)),
- constraints: There are four binary constraints, expressing the fact that a, b, c, and d make up a straight rectangle:

```
x(a) = x(c)
and y(a) = y(b)
and x(b) = x(d)
and y(d) = y(c)
```

Let `S` be the set of solutions for CSP1. The size of `S` is n! / (n-4)!. This size is in $O(n^4)$, to be compared with $O(n^8)$ in the naive version of the second solution.

CSP P2
• variables: r_1, r_2 , with domain `S`.
• constraint: one binary constraint expressing the fact that r1 and r2 do not intersect, and using method `intersects` defined in class `Rectangle`.

Finally, CSP2 is solved, yielding all pairs of rectangles satisfying (C). Summing up the two phases, the total complexity of the problem using this approach is therefore in $O(n^4)$, with only binary constraints.

4.4 Theoretical Complexity of the approach

For a detailed survey of this two-phase resolution, see [Roy and Pachet 1997a].
The theoretical complexity of our approach can be computed as follows.
Following the attribute-based approach, there are 8 variables, point, whose domain contains n^2-n elements. Since the treatment of a constraint consists in computing its Cartesian product, the total complexity is $(n^2-n)^8$, that n^{16}.
In the class-based approach, the first problem contains two sets of 4 variables. The domains of these variables contain n^2-n elements. There is one 4-ary constraint within each group. Therefore, enumerating the rectangles of the second problem costs $2.(n^2-n)^4$. The resulting problem contains two variables of size *n! /(n-4)!*, linked together by a binary constraint. The complexity of the second problem resolution is therefore $(n!/(n-4)!)^2$, that is n^8. Finally, the total complexity of this approach is n^8. This is to be compared to the complexity of the resolution following the first approach, which is n^{16}.

This result is particularly interesting in this case because of the scarcity possible “rectangles with their sides parallel to the axis”. Indeed, the number of possible such rectangles is the square root of the total number of tuples of 4 points.

However, the general saving of the approach can be evaluated as follows: let A be a set of n atomic objects. The size of the set of all n-uplets of A is $card(A)^n$. the set of composite objects can be seen as the image of A by a composition mapping, called c.

If one considers a problem with two composite objects, and thus 2.n atomic objects. On the one hand, the complexity of the attribute-based approach is $(Card(A)^n)^2$. On the other hand, the complexity of the class-based approach is $(Card(c(A^n)))^2$. If r denotes the ratio $Card(A)^n / Card(c(A^n))$, this quantity measures the “density” of composite object in the space of every n-uplets. Since, the ratio between the complexities of the two approaches is r^2, the scarcer are composite objects, the more efficient is the class-based-approach.

The experiment with the rectangle problem are in accordance with the previous remarks:

For n = 4, there are 6 possible straight rectangles, and 6^2 couples of rectangles. There are 14 solutions of (P). It took 15 seconds to enumerate all solutions using the attribute-based approach, and 9ms using the second approach with our separation scheme.

For n = 6, there are 90 possible straight rectangles, and 90^2 couples of rectangles. There are 4090 solutions of (P). It took 5 minutes to enumerate all solutions using the first approach, and 2 seconds using the second approach with our separation scheme.

4.5 Generalization

The resolution scheme presented here can be generalized to problems involving more than two levels of composition. For instance, in the problem (P) there are two composition levels if we consider the points as atomic objects, but there are three levels (coordinates, points and rectangles) if we consider points as aggregate structures.

In general, problems may include an arbitrary number, say c, of composition levels. In this case we claim that designing and solving the problem into c different phases, each phase corresponding to a particular level, leads to considerable improvements:

- When solving a CSP with the first method, the variable ordering heuristic is fixed for the duration of the whole resolution. With our separation scheme, we can use a specific variable ordering heuristic for each phase of the resolution, and more generally exploit available meta knowledge specific to each phase.
- We saw that this separation allows to easily reuse existing classes without any modification.
- The amount of constraints and variables needed, when designing a CSP with our separation scheme, is much less important than with the first solution.

4.6 Related works

This work is related to problem reformulation based on clustering of variables. For instance, the paper by (XXxIJCAI sur le partitionnement de constraint problems) somehow addresses the same problem, but follows a bottom-up approach, starting from a flatted formulation, and trying to build clusters that minimize the number of search nodes in the resolution. We follow a top-down approach, that may not be as optimal, but which is interesting because it is based on meaningful abstractions, for which 1) models can exist already, and2) implementations can be reused.

5. Formulating the AHP using the class-based approach

These remarks led us to reconsider the AHP problem in the light of the class-based approach for object + constraint problems, that is, with a reverse viewpoint from our predecessors. Rather than “starting from the constraints”, and devising object structures that fit well with the constraints, we “start from the objects”, and fit the constraints to them. Indeed, a lot of properties of the domain objects may be more naturally described as methods instead of constraints.

To do so, we propose to reuse a fully-fledged object-oriented library, the MusES system [Pachet 1994], which contains a set of around 90 classes that represents the basic elements of harmony, such as notes, intervals, scales and chords. The constrained variables of our problem have musical objects, provided by MusES, in their domain (e.g. notes, chords, intervals). The constraints hold directly on these high-level objects, using the methods defined in the corresponding classes. These constraints are instances of block and perform BACKTALK constraints, introduced in Section **Erreur ! Source du renvoi introuvable.**.
The main idea here, is to consider high-level objects of the problem, namely chords, as possible values for constrained variables. In other words, the statement of the problem uses constrained variables whose domains contain fully-fledged chords, which are instances of MusES class `Chord`. As explained in Section **Erreur ! Source du renvoi introuvable.**, to do so, we need to compute all the possible chords to put in the domains. The problem here is that no chord are given with the problem's definition. The solution to this problem is to consider all possible chords of harmony as possible values. Unfortunately, this is not realistic since the number of possible chords is enormous.

In accordance with the analogy given above, we apply our resolution scheme to AHP as follows, by breaking the problem into two parts:

1) Management of the constraints at the note level only.
Input of the n-note melody and creation of a CSP with only constraints on notes.
Arc-consistency is applied to reduce the domains of the note variables.

2) Management of the constraints at the chord level.
Computation of the concrete instances representing all the possible chords under each note.
Creation of a second CSP including only the higher level objects, i.e. chords variables and constraints involving chords.
Arc-consistence is applied, followed by the enumeration of solutions with forward-checking.

In this scheme, given a n-note melody, the total CSP contains (4*n) variables for the notes plus n variables for the chords, which are handled in the second phase. The results are given in Figure 7. As we can see, we are an order of magnitude faster than previous approaches. The memory needed is not significant.

1
- Input n-note melody
- Build CSP with constraints on notes (4n variables)
- Arc-consistency
- Computation of chord domains

2
- Creation of second CSP with n constrained variables (chords)
- Arc-consistency
- Enumeration by forward-checking

Figure 6. Diagram of the architecture.

5.1 Reusability

As a consequence of reifying chords as values for variables, the resulting system is very much understandable, because constraints can be stated using directly the language of chords. For instance, the rule forbidding parallel fifth is simply defined by the following BACKTALK expression:

```
BTConstraint
   on: chord and: nextChord
   block: [:c1 :c2 | c1 (hasParallelFifthWith: c2) not]
```

Moreover, the problem to solve is much smaller, since for a *N*-note melody, only $5 \times N$ constrained variables are created (to be compared to the $126 \times N - 28$ variables created in Ballesta's attribute-based system). This results in improving efficiency in a dramatic way, as shown in Figure 4, which gives the performance of our application compared with the systems of Ballesta and Tsang. Ovan's work is not reported here since it addresses a simpler problem (two-voice harmony exercises instead of four voice).

5.2 Results

	11 notes	12 notes	16 notes
Tsang (CLP)	60 sec.	?	?
Ballesta (ILOGSOLVER)	?	180 sec.	240 sec.
BACKTALK + MusES	1 sec.	1 sec.	1.5 sec.

Figure 4. Comparing the performance of our solution (BACKTALK + MusES) with previous approaches using constraint satisfaction techniques to solve the AHP.

6. Other applications

The domain of Multimedia often includes domains which are naturally structured by part-whole relations. We outline here a project in progress concerning the automatic generation of music recitals, for which or approach was used successfully.

In this system, the aim is to generate automatically a recital, which is a sequence of musical pieces taken from a given repertoire. The recital must satisfy certain conditions, expressed as constraints. In our project, the pieces are taken from a repertoire of French Baroque Harpsichord Music, and the conditions come from studies by well-known musicologists (BukovserXXX). Typical constraints are “two contiguous pieces should be of a different type”, or “all pieces should be in the same key”. However, it appears that recitals are naturally structured into so-called “blocks” : the introductory block, an optional block and a conclusive block. Additional conditions must hold on these blocks. For instance, if there is an optional block, there has to be a conclusive block. Other constraints hold on pieces within a block. For instance, the introductory block must begin by an “allemande” or a “pavane”, and may include a gigue, in between the first and third pieces.

Although several works were devoted to designing specialized global constraints for building constrained sequences (Baptiste et al. 94), (Beldiceanu XXX???), our approach can be used here profitably for reducing the search space, without requiring a specialized algorithm.

It is clear that the nature of the problem makes it fit naturally in our scheme. More precisely, we identify a part-whole hierarchy Recital/Block/Piece, and split the constraint set into two parts: constraints holding only on pieces, and constraints holding only on blocks. The resulting system is then efficient enough to handle large repertoires of musical pieces, which would be otherwise impracticable using standard CSP techniques.

7. Conclusion

8. References

Amarel, S. (1966). “Comments on the Mechanization of Creative Processes.” *IEEE Spectrum*.

Bukofzer, M. F. (1947) Music in the Baroque Era, from Monteverdi to Bach. Norton & Company.

Ebcioglu, K. (1992). An Expert System for Harmonizing Chorales in the Style of J.-S. Bach. Understanding Music with AI: Perspectives on Music Cognition. M. Balaban, K. Ebicioglu and O. Laske, AAAI Press: 294-333.

Ebcioglu, K. (1993). An Expert System for Harmonizing Four-Part Chorales. Machine Models of Music. S. M. Schwanauer and D. A. Levitt, MIT Press: 385-401.

Steels, L. (1979). Reasoning modeled as a Society of Communicating Experts, MIT AI Lab.
Steels, L. (1986). Learning the craft of musical composition. ICMC, The Hague (Netherlands).

Courtot, F. (1990). A Constraint Based Logic Program for Generating Polyphonies. ICMC 90, Glasgow.

Ovans, R. and R. Davison (1992). An Interactive Constraint-Based Expert Assistant for Music Composition. Ninth Canadian Conference on Artificial Intelligence, University of British Columbia, Vancouver, pp. 76-81.

Ovans, R. (1992). Efficient Music Composition via Consistency techniques, Simon Fraser University, Canada.

Tsang, C. P. and M. Aitken (1991). Harmonizing music as a discipline of constraint logic programming. ICMC '91, Montréal.

Ballesta, P. (1994). Contraintes et objets: clefs de voûte d'un outil d'aide à la composition. PhD. Thesis, Université du Maine.
Also same title, chapter in “Journées d’Informatique Musicale 94-96”, Hermes Eds, 1998, to appear.

Henz, M., S. Lauer, et al. (1996). CompoZe- Intention-Based Music Composition Through Constraint Programming. 8th IEEE International Conference on Tools with AI, Toulouse (France).

P. Baptiste, B. Legeard, H. Zidoum (1994) Sequence Constraint Solving in Constraint Logic Programming", , International Coference on Logic Programming - Workshop on Logic programming with Sets, 6th IEEE Int. Conf. on Tools for Artificial Intelligence, TAI'94, NewOrleans, USA, November 6-9, 1994, pp. 804-807, IEEE Computer Society Press.