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

François Pachet[1], Pierre Roy[2]

Abstract. We address the issue of formulating constraint problems in 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 (CSP) have been increasingly used for solving difficult combinatorial problems recently, thanks to the availability of efficient algorithms and data structures. CSPs are attracting because they allow problems to be stated in a simple and declarative way, as 1) a set of variables, having predetermined finite domains, and 2) a set of constraints, usually taken off-the-shelf from a constraint library, which hold their own solving mechanism.

Today, most of the problems solved by these techniques traditionally belong to the field of operational research, and combinatorial optimization. These problems are usually formulated using some kind of mathematical representation, such as graph theory. A typical consequence of this formulation is that the domains of constrained variables are "flattened out", and the whole problem is stated in terms of atomic values, such as integer or real numbers. Somehow, the semantics of the domains is integrally represented by the constraints, and the domains of variables have no direct interpretation. This approach is efficient in cases when the resulting constraint systems contains only well-known constraints, such as basic arithmetic constraints or combinations thereof, or so-called *global constraints*, for which generic, parameterized solving algorithms were developed [Beldiceanu and Contejean 94]. This typically occurs for problems which have already been the subject of formalization in fields such as operational research: scheduling, layout cutting, etc.

We believe CSP techniques can be successfully applied to many other fields, where 1) problems have not been formalized mathematically, and 2) domains are naturally structured. This is typically the case in multimedia, where there is a growing pool of new problems which call for efficient solving techniques, and in which the underlying ontologies are not easily reduced to mathematical entities such as integer or real numbers. However, applying CSP techniques in this context raises an important formulation issue, by definition, since the reduction of problems to a flattened representation is either impossible or unnatural. In this context, we are interested in the formulation of constraint problems which allows to 1) take into account underlying ontologies of structured objects, and 2) give rise to efficient solving by traditional CSP techniques.

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 2 we will describe precisely the automatic harmonization problem, and review preceding approaches for solving it. In Section 3 we will describe our approach, based on a formulation of the problem in a non-flattened space, and compare it to the equivalent flattened formulation, including an evaluation of its theoretical complexity.

[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

We will apply the approach on the harmonization problem in Section 4, and conclude on the generality of our proposal, and other applications in progress.

2. THE AUTOMATIC HARMONIZATION PROBLEM

This section introduces the harmonization problem, and outlines previous attempts at solving it automatically.

2.1 Definition

The automatic harmonization problem (AHP) 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). Rules of harmony have constantly evolved during history, but most treatises of harmony, such as [Schoenberg 78] describe rules corresponding to the baroque and classical eras. However, the set of rules can be easily modified to realize harmony exercise in a style corresponding to a different period.

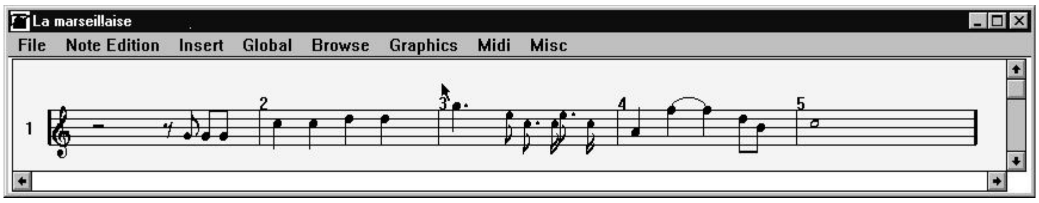

Figure 1. An initial melody to harmonize (the beginning of the French national anthem, 18 notes). Rhythm is not significant here.

A standard exercise is to harmonize a given melody into a four-voice harmonization (see Figure 3 for a solution of the melody of Figure 1). 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*. In these exercises, rhythm is not taken into account, and each note of the melody is harmonized by a corresponding chord.

Constraints on sequences of chords are the most famous and important in our context. These constraints are important because they are explicitly stated as holding on "chords", where chords are sets of simultaneously occurring notes. This notion of chord introduces a natural part-whole relationship between chords and notes in the domain. Moreover, chords are not only sets of notes, but have various properties: a *name*, computed from the intervals between their notes, a *degree* representing a kind of abstraction with regards to a particular scale, and so forth. A typical constraint on sequences of chords is the parallel fifth constraint which states that: "parallel fifth between two successive chords are forbidden", i.e. if a chord contains a fifth interval between two of its notes, then the corresponding notes in the successive chord should not make up also a fifth interval, as illustrated in Figure 2. Another similar constraint is the different degree constraint: "two successive chords should have different degrees".

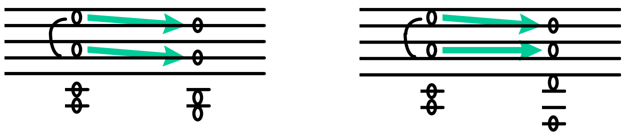

Figure 2. The parallel fifth constraint, holding on two successive chords. On the left, two chords which violate the constraint. On the right, two chords which satisfy the constraint.

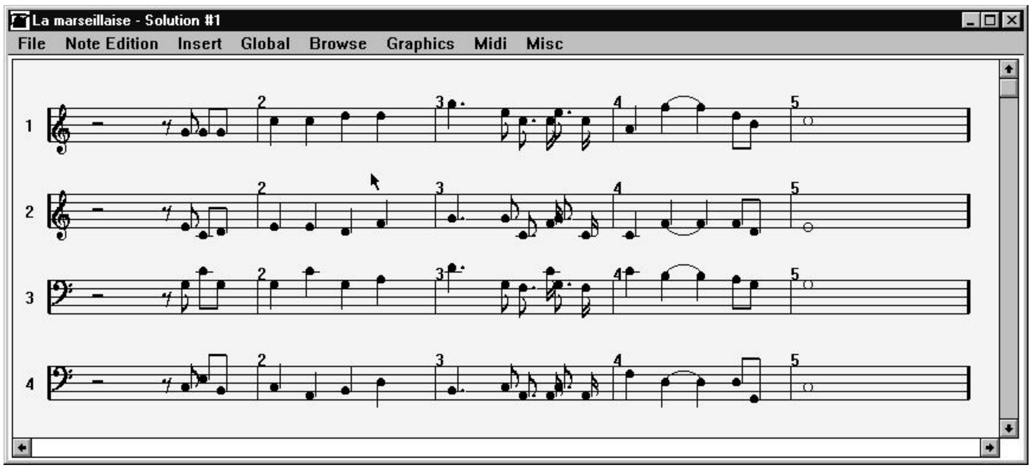

Figure 3. A solution for the initial melody of Figure 1.

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. It is therefore not surprising that several studies have proposed various solutions to solve the automatic harmonization problem using constraints. We will review them in the following section, and outline their main characteristics. It is to be noted that a human student typically solves these kinds of exercises quickly, with little or no “backtracking”. The problem is therefore a challenge for artificial intelligence, because it is both difficult from a combinatorial viewpoint (See Section 2.2), and easy for humans.

2.2 Previous Approaches to the AHP

The first system historically, which solved the AHP is probably Ebcioglu’s Choral system [Ebcioglu 92]. Choral’s objective is to generate fully-fledged chorales in the style of J.S. Bach from scratch. To do so, Ebcioglu devised an ad hoc constraint logic programming language (BSL), based on production rules integrated in a backjumping-based algorithm. Choral includes several interacting modules for melody generation, harmonization and analysis, as well as modules to consider stylistic constraints. From a technical point of view, constraints are used mostly *passively*, to reject solutions produced by production rules. It is therefore difficult to assess the quality of the harmonization module only.

Levitt [Levitt 93] proposed a fully-fledged system for producing complete orchestrations according to various musical styles, such as ragtime, using constraints, but used value *propagation* techniques, in a perturbation model [Borning 81], rather than satisfaction techniques, and was therefore unable to solve our problem. In the same spirit, Steels approach [Steels 86] 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 combine brute exploration with heuristic search, an important concern of CSP in the large. The use of arc consistency (a widespread technique in constraint satisfaction), for musical harmonization was first advocated by Ovans in [Ovans and Davison 92]. He showed experimentally that techniques based on arc consistency were more adapted and efficient than techniques based on earlier techniques, such as pure backjumping. However, he used a flat representation of the harmonization problem, and addressed only the problem of two-voice harmonization. In addition, his results, although better than his predecessor’s, were still unsatisfactory.

Tsang proposed a solution of the four-voice harmonization problem using constraint logic programming techniques [Tsang and Aitken 91]. Although the proposed system was operational, the authors still concluded on the impracticability of the approach, for efficiency reasons: it took more than 5 minutes, on a SunSparc 1 workstation, and required 70 Megabytes of memory to solve the AHP on an 11-note melody. They employed a flattened representation of the musical domain: constraints were stated on “notes” variables only, themselves represented as mere integers.

More recently, Henz addressed the harmonization problem from a different viewpoint, emphasizing the need for developing a plan, prior to the resolution [Henz et al. 96] . 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 133 MHz PC). The resulting system is interesting from a musical point of view because it allows users to effectively test harmonization plan interactively. However, it is less relevant from the constraint satisfaction point of view, since the constraint problem addressed is actually a very simple combinatorial problem (harmonization with chord degrees given), which can be solved easily by standard CSP technique (the Oz system [Smolka 94] in this case).

The system proposed by Ballesta in [Ballesta 98] is probably the most complete attempt at solving the four-voice harmonization problem using recent constraint techniques (a Lisp version of the Ilog Solver system [Puget 95]). The approach taken by Ballesta, however, is basically the same than in [Tsang and Aitken 91]: the constraints are all stated on a flattened representation of the musical objects (notes and chords). For instance, in [Ballesta 98], 12 attributes are defined to represent one interval, such as the *name* of the interval (e.g. diminished fourth), its *degree*, and its two *extremities*. Nine constraints are then introduced to state the relations that hold between the various attributes of class Interval. For instance one 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 are defined using a low-level language (arithmetic), thus require a translation of harmonic and melodic properties in terms of integers. For instance, the parallel fifth constraint is represented as follows (actually a set of six 4-ary constraints):

```
parallel-fifth(n1, n2, n3, n4, n'1, n'2, n'3, n'4) ⇔
if fifth(n1, n2) then ¬fifth(n'1, n'2)
if fifth(n1, n3) then ¬fifth(n'1, n'3)
if fifth(n1, n4) then ¬fifth(n'1, n'4)
if fifth(n2, n3) then ¬fifth(n'2, n'3)
if fifth(n2, n4) then ¬fifth(n'2, n'4)
if fifth(n3, n4) then ¬fifth(n'3, n'4)
```

Figure 4. The constraint corresponding to the parallel fifth rule, as expressed in the flattened formulations, e.g. [Ovans and Davison 92] and [Ballesta 98]. The same constraint expressed using structured domains is given in Section 4.

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. To solve the AHP on a *n*-note melody, his system 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. Although the complexity of Ballesta's approach is difficult to evaluate since the system uses special classes of constraints, one can approximate the search space by 14^3n, where n is the number of notes, $3.n$ is the number of constrained note variables, and 14 is the average number of possible notes for each variable. If n = 15 notes (typical case), the search space is then $14^{45} = 10^{52}$.

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 reflect 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 being made at the chord level (e.g. the parallel fifth constraint).

Our basic idea is to exploit the natural structuring of objects in this problem: chords. As we saw above, chords can be defined 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 the next section, we will propose an integration of this part-whole hierarchy in the formulation of the constraint problem, and show how it allows the statement of the problem in a simpler way, while ensuring a better efficiency.

3. EXPLOITING NATURAL PART-WHOLE HIERARCHIES IN FORMULATING CSPS

In order to formulate precisely our 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.

3.1 Definition of a CSP

Let us first recall that a CSP is defined by 1) a set V of variables, each variable having a finite domain, and 2) a set of constraints. Constraints are defined either in intension by a predicate holding on some variables, or in extension by the set of consistent tuples. A solution of a CSP is an instantiation of all variables that satisfies simultaneously all the constraints.

3.2 Two Main Designs for CSP and Structured Domains

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

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 rectilinear rectangles.
C4 - The two rectangles do not intersect.

Figure 5 shows a solution of (P) when n = 10.

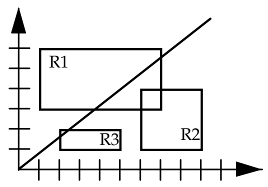

Figure 5. A solution of problem (P) is a pair of rectangles satisfying the set of constraints (C). The pairs of rectangles {R1, R3} and {R2, R3} are possible solutions; pair {R1, R2} is not. In a space with coordinate ranging from 1 to 6, there are 90 solutions.

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, considering here points as atomic entities. In this representation, the constraints are stated entirely in terms of point variables. Reach of these variable has a domain containing the set al all points with integer coordinates, where each point is itself a couple of integers (*x*, *y*). The only variables in this formulation are the point variables; the coordinates are not variables.

The three first constraints would be stated as follows, where a, b, c, d (resp. a', b', c', d') are the variables representing the point coordinates of Rect1 (resp. Rect2), and x and y are the functions yielding the x and y coordinate of the points (see Figure 6).

C1: unary constraints on each variable: dom(a) = dom(b) = … = {all points with integer coordinates in {1.. n}}

C2: unary constraints on each variable: dom(a) = dom(b) = … = {all points with different coordinates}

C3: sets of four binary constraints for stating that each rectangle is rectilinear:

x(a) = x(c); y(a) = y(b); x(b) = x(d); y(c) = y(d)

x(a’) = x(c’); y(a’) = y(b’); x(b’) = x(d’); y(c’) = y(d’)

The most important constraint, C4, is a 8-ary constraint, stated as follows:

```
x(b) < x(a')       "Rect1 on the left of Rect2"
or x(a) > x(b')    "Rect1 on the right of Rect2"
or y(c) > y(a')    "Rect1 above Rect2"
or y(a) < y(c')    "Rect1 below Rect2"
```

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).

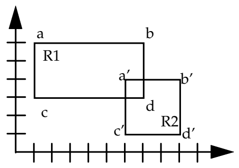

Figure 6. Stating constraint C4 in terms of points variables.

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 points, a natural solution consists in specifying a problem with *both* "rectangle variables" (i.e. variables whose domain is collection of rectangles) and point variables. The main interest of this solution is to allow the statement of constraints in a more natural manner, and to exploit the natural part-whole relation between rectangles and points.

In this formulation, constraints C1, C2, and C3 would be stated similarly, but constraint C4 would then be stated as the following binary constraint:

```
not (intersects (Rect1, Rect2))
```

In this solution, there would be:

- 8 point variables as in the first solution,
- 2 additional 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).
- the same constraints C1, C2, and C3 than in the first formulation
- 1 binary constraint, for C4.

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! In the second solution, constraints are stated on higher-level objects (rectangles). Also, the constraints involve less variables (the arity of constraints is a crucial parameter for performance). In our example, C4 has eight variables in the first solution, and only two in the second one.

Of course, this formulation raises a problem: how to build rectangles to create the domains for the two additional rectangle variables? Creating these domains naively would involve the creation of a lot of objects: in our case it is roughly equal to the Cartesian product of the 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 7. The first one corresponds to a problem with partially instantiated objects, i.e. objects whose attributes are constrained variables. In the second one, constraints are stated between fully instantiated objects.

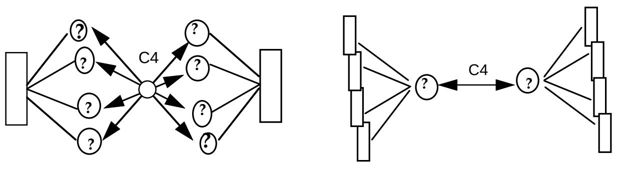

Figure 7. The two design approaches in designing a CSP + part-whole hierarchy problem. Solution 1 is on the left, 2 on the right.

3.3 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 between domain values yields a natural partition of the set of variables and of the set of constraints. For the sake of simplicity, we will consider here only “one-level” part-whole relationships. The generalization to arbitrary levels is easy.

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 v1 is a ”part of” v2*

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:

$$C = C1 \cup C2 \cup C_{mixed} \text{ with } C1 \cap C2 = \varnothing \text{ and } C1 \cap C_{mixed} = \varnothing \text{ and } C2 \cap C_{mixed} = \varnothing$$

where the subset *C1* (resp. *C2*), 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*. The strategy is the following: in a first step, we consider only atomic objects of the problem and the constraints involving them, i.e. *V1* and *C1*. We then apply domain reduction to remove inconsistent values from this initial problem values. We then compute all the higher level objects that can be made up with the remaining atomic values. These objects make up the domains of the variables in *V2*. Finally, we can solve the full problem as defined in our second formulation.

The strategy can be summarized as follows:

First phase

We first create a CSP *P1*, whose variable set is *V1*, and whose constraint set is *C1*. We apply a consistency algorithm to *P1*, in order to reduce the size of the domains of its variables as much as possible.

Intermediate phase

The intermediate phase consists in computing the domain of every *v2* in *V2*. These domains are computed with respect to the reduced domains of each *v1* that satisfies *v1 R v2*. This computation is done by simply enumerating, for each *v2*, the Cartesian product of its parts, i.e. every *v1* such as *v1 R v2*.

Second phase

We then create a CSP *P2*, whose variable set is $V{=}V1{\cup}V2$, and whose constraint set is $C1{\cup}C2{\cup}C_{mixed}$. The domains of variables in *V2* were reduced during the intermediate phase.

Notice that in the general case, constraints in *C1* have to be considered in the second phase, because arc consistency has not necessarily ensured global consistency. For this reason, it could seem that our decomposition is useless. However, it is not the case, and the main result of our approach is that the cost contribution of these constraints in the second phase is small compared to the cost of *C2* in the flattened formulation.

We will now evaluate the complexity of this approach, and then apply it to the rectangle problem and the harmonization problem.

3.4 Theoretical Complexity of the Approach

The contribution of our 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 $\text{card}(A)^n$. The set of composite objects can be seen as the image of A by a composition mapping, called *c*.

Let us consider a problem with two composite objects, and thus *$2.n$* atomic objects. Let us then suppose one global "arbitrary" constraint, i.e. for which there is no particular filtering method available.

The complexities of the two approaches can be computed as follows:

Approach #1

In the first approach, the composition constraints hold on at most n atomic variables. Their complexity is therefore bounded by $Card(A)^n$. The global constraint holds in $2.n$ atomic variables, whose domains is A. Its complexity is $(Card(A)^n)^2$. The total complexity of the flattened approach is $Card(A)^{2n}$.

Approach #2

In this approach the complexity of the first phase is the same as above, i.e. $Card(A)^n$. The intermediate phase consists in computing the Cartesian product of n variables whose reduced domains are bounded by $Card(A)$: its complexity is bounded by $2.Card(A)^n$. In this approach, the global constraint holds on two composite variables whose domains contains $c(A^n)$ elements. Its complexity is $(Card(c(A^n)))^2$. The total complexity of the second approach is thus $(Card(c(A^n)))^2 + 2.Card(A)^n$.

In general, $Card(A)^n$ is negligible before $(Card(c(A^n)))^2$, so the total complexity may be approximated by $(Card(c(A^n)))^2$.

Let r be the ratio $Card(A)^n / Card(c(A^n))$. r measures the "density" of composite objects in the space of all n-uplets. Since the ratio between the complexities of the two approaches is r^2, we deduce that the scarcer are composite objects, the more efficient is our approach.

In the case of rectangles, the scheme instantiates as follows: the number of 4-plets is n^8, while the number of rectilinear rectangles is $n!/(n-4)!$, which is approximated by n^4. Therefore the ratio $r = n^4$.

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 themselves as structures of two integers.

3.5 Related Works

Our work is related to the general issue of problem reformulation. The impact of formulation of CSP on their resolution has already been emphasized by [Nadel 90]. Nadel identified several equivalent formulation of the n-queen problem, including "dual" approaches, in which constraints and variables are somewhat swapped. In our case, we do not try to find a "dual" representation space, but rather to build a representation space in which structures are made explicit, and considered themselves as variables. This idea is similar to the ideas underlying the Robin Chess program [Pitrat 77]: these representation spaces are interesting because 1) they are much smaller than the initial flattened space, and 2) knowledge can be expressed directly in terms of these structures and exploited by the solver (chord structures for music; plans in the case of chess).

For instance, CSP decomposition, see e.g. [Weigel and Faltings 97], somehow address the same problem, but follow 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, and 2) implementations can be reused.

Another way to reduce the complexity of CSPs is by exploiting topologies of *already flattened* CSPs, such as tree-structured [Bayardo and Miranker 94]. In our case this is not applicable because 1) we precisely do not want to build the flattened formulation and 2) If we were able to produce this formulation, it would not exhibit a tree-like topology, since there remains constraints holding "across" structured objects (for instance constraints between consecutive notes, which create cycles).

4. Formulating the AHP Using the Class-Based Approach

The preceding approach and results can now be directly applied to the AHP, by identifying the natural part-whole relation between chord objects and notes.

To do so, we reused a fully-fledged object-oriented library, the MusES system [Pachet et al. 96], which contains a set of around 100 classes that represents the basic elements of harmony, such as notes, intervals, scales and chords. Constraints are stated and solved using the BackTalk system [Roy and Pachet 97], designed to allow constraints to hold directly on objects using arbitrary methods defined in their classes.

Using our approach, given a n-note melody, the total CSP contains *3.n* variables for the notes plus n variables for the chords (handled only in the second phase). Some constraints are stated both at the note level (C1), as in the flattened approaches. Constraints on chords (C2) are directly expressed on chord variables. For instance, the famous parallel fifth constraint is expressed as follows:

```
Not (hasParallelFifth (chord1, chord2))
```

where `chord1` an `chord2` are two chord variables, and `hasParallelFifth` is a standard predicate (a method in class `Chord`).

Solutions for melodies of around 15 notes are found almost instantaneously (less than 1 second on a Pentium PC, using the Smalltalk language).

Comparing precisely our approach to the preceding ones is difficult to do, because the previous authors did not solve the same problem, the melodies were different, the solvers were different, as well as the computers used. Ovans addresses a much simpler problem than ours: unfigured two-voice harmony exercises instead of four-voice. The comparison is therefore not relevant. Similarly, Henz's system solves a harmonization with a predetermined harmonization plan, which is also a simpler problem, but not enough information is available to compare it in detail with ours. We give here some precise elements for assessing our approach and comparing to Tsang's and Ballesta's.

• Comparison with Tsang's approach.

Tsang is the only preceding approach solving the full 4-voice unfigured harmonization problem. Tsang gives only one sample melody. The comparison on this melody is given in Figure 8. There is a 1000 factor between these two figures. If we consider that there is a factor of 100 between our PC and the computer used by Tsang, our result is 10 times better, notwithstanding the memory space which is much less in our case. However, this comparison is limited since there is only one melody, and we do not have much information on the performance of the solver used by Tsang.

	Tsang's 11 note melody
Tsang (CLP)	5 minutes on a SunSparc 1 workstation + 70 Megabyte memory space
BACKTALK + MusES	0.34 secs on a Pentium 300 MHz computer + 1.6 Megas memory (54 backtracks)

Figure 8. Comparison between our approach and Tsang's on a 11-note melody.

• Comparison with Ballesta's approach.

Ballesta solves the figured bass problem, which is much simpler than the unfigured bass one. Indeed, figure information limits drastically the number of possible chords (there are in average 8 possible chords for a given figure). Solving the figured bass problems is, in our case, even simpler than solving only C2 alone (the second phase), since the domains of chord variables are already given. Some of Ballesta's melodies are even very difficult to solve without the figure information. Our system allows to solve all the melodies of Ballesta, without the figures, and much faster.

To give more elements of comparison, we took a bass line proposed by Ballesta (see Figure 9), and solved the unfigured bass problem using our system on segments of the bass line of increasing length (2 to 14 notes). The results are given in Figure 10. We give the CPU and number of backtracks for the computation of the first solution, for both systems (Ballesta's and ours). The difference in performance between the Sparc 10 and the Pentium 300MHz can be estimated to be a factor of 10. This yields a factor of 50 in favor of our approach, on a much harder problem.

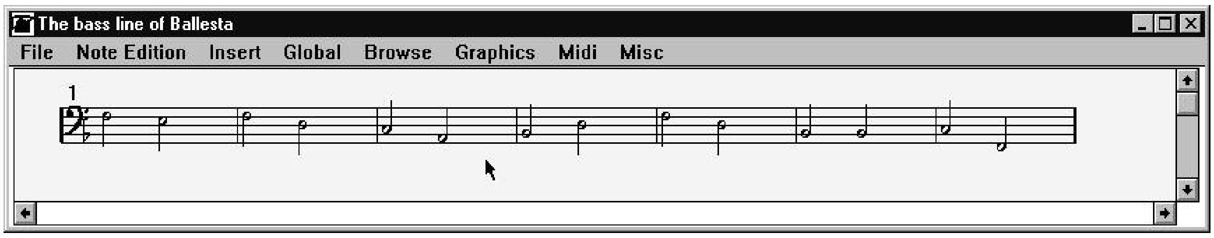
The bass line of Ballesta
File Note Edition Insert Global Browse Graphics Midi Misc
1

Figure 9. The initial bass line for our benchmarks.

Note that we also applied our system to the figured bass problem, as Ballesta, and also obtained better results. However, the improvement of performance is not so significant in the context of this paper, since the figured bass problem involves a simplified part-whole relation. These results are not reported here for reasons of space limitations.

Note also that we have not reported the results for finding *all* solutions of given bass or melody lines. This is because the solutions sets are different: the absence of figure information increases dramatically the number of solutions by several orders of magnitude, so this comparison is not relevant.

	Ballesta, Figured Bass, 1st Solution	Our approach, Unfigured Bass, first Solution				
length	CPU (in seconds)	CPU, Phase #1	CPU, Phase #2	CPU, Phase #3	CPU Total	fails
2	9	0,007	0,042	0,013	0,062	1
3	45	0,007	0,029	0,031	0,067	6
4	120	0,006	0,064	0,063	0,133	7
5	90	0,007	0,066	0,167	0,24	36
6	160	0,008	0,069	0,17	0,247	36
7	180	0,008	0,085	0,18	0,273	36
8	130	0,022	0,061	0,18	0,263	36
10	170	0,009	0,085	0,207	0,301	40
12	191	0,009	0,102	0,24	0,351	41
14	184	0,011	0,108	0,28	0,399	43

Figure 10. Comparison between the system of Ballesta and ours. The table shows the resolution of a figured bass exercise in Ballesta's approach, for increasingly long bass lines. For our approach, we give additionally the number of backtracks. Note that our system solves a harder problem because figures are not given.

5. 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 [Bukofzer 47]. 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], 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.

6. CONCLUSION

This paper addresses the issue of formulating constraint problems in structured domains. We propose to explicitly take into account part-whole hierarchies, when they occur, at *the problem formulation level*, rather than during execution. We show that these relations allow to state the problem in a representation space in which domains are smaller, thus lead to gains in efficiency. Additionally, the proposed approach allows the statement of problems in a more natural way than using a classical, flattened representation of domains. In this respect, our approach fits in the general issue of problem reformulation [Amarel 66], where the aim is to identify representation spaces where either domains are smaller or more knowledge is available.

REFERENCES

[Amarel 66] S. Amarel. (1966) Comments on the Mechanization of Creative Processes. *IEEE Spectrum*.
[Ballesta 98] Ph. Ballesta. (1998) Contraintes et objets: clefs de voûte d'un outil d'aide à la composition. Chapter in “Recherches et applications en informatique musicale”, Hermès Eds, Paris, France.
[Baptiste et al. 94] B. Baptiste, H. Legeard, and H. Zidoum. (1994) Sequence Constraint Solving in Constraint Logic Programming. International Conference on Logic Programming - Workshop on Logic programming with Sets, 6th IEEE Int. Conf. on Tools for Artificial Intelligence, TAI'94, NewOrleans, USA, 804-807, IEEE Computer Society Press.
[Bayardo and Miranker 94] R.J. Bayardo, and D. P. Miranker. (1994) An Optimal Backtrack Algorithm for Tree-Structured Constraint Satisfaction Problems. Artificial Intelligence 71: 159-181
[Beldiceanu and Contejean 94] N. Beldiceanu and E. Contejean. (1994) Introducing global constraints in CHIP. Math. Comput. Modelling. 20(12):97-123
[Borning 81] A. Borning. (1981) The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. *ACM transaction on Programming Languages and Systems*, 3(4), 353-387
[Bukofzer 47] M. F. Bukofzer. (1947) Music in the Baroque Era, from Monteverdi to Bach. Norton & Company
[Ebciolgu 92] K. Ebcioglu. (1992) An Expert System for Harmonizing Chorales in the Style of J.-S. Bach. Understanding Music with AI: Perspectives on Music Cognition. AAAI Press: 294-333
[Henz et al. 1996] M. Henz, S. Lauer and D. Zimmermann. (1996) CompoZe- Intention-Based Music Composition Through Constraint Programming. 8th IEEE International Conference on Tools with AI, Toulouse (France)
[Levitt 93] D.A. Levitt. (1993) A Representation of Musical Dialects. Machine Models of Music. S. M. Schwanauer and D. A. Levitt, MIT Press**:** 456-469
[Nadel 90] B. Nadel (1990). The representation selection of constraint satisfaction: a case study using n queens. IEEE Expert, pp. 16-23
[Ovans and Davison 92] R. Ovans and R. Davison. (1992) An Interactive Constraint-Based Expert Assistant for Music Composition. Ninth Canadian Conference on Artificial Intelligence, University of British Columbia, Vancouver, 76-81, 1992
[Pachet et al. 96] F. Pachet, G. Ramalho, and J. Carrive. (1996) Representing temporal musical objects and reasoning in the MusES system. *Journal of New Music Research*, 25(3): 252-275
[Pitrat 77] J. Pitrat. (1977). “A chess combination program which uses plans.” *Artificial Intelligence*, 8, 275-321.
[Puget 95] J.-F. Puget. (1995) Beyond the Glass Box: Coonstraints and Objects. ILPS’95, Portland, Oregon, pp. 513-527
[Roy and Pachet 97] P. Roy and F. Pachet. (1997) Reifying Constraint Satisfaction in Smalltalk. *Journal of Object-Oriented Programming*, 10(4):51-63
[Schoenberg 78] A. Schoenberg. (1978) *Theory of Harmony*, University of California Press, Berkeley
[Smolka 94] G. Smolka. (1994) The Oz programming model. In J. van Leeuwen, editor, Computer Science Today, LNCS, vol. 1000, pp. 324-343
[Steels 86] L. Steels. (1986) Learning the craft of musical composition. ICMC, The Hague (Netherlands), 86
[Tsang and Aitken 91] C. P. Tsang, and M. Aitken. (1991) Harmonizing music as a discipline of constraint logic programming. ICMC '91, Montréal
[Weigel and Faltings 97] R. Weigel and B. V. Faltings. (1997) Structuring techniques for constraint satisfaction problems. Proc. of IJCAI’97, Nagoya, 418-423