Aperiodic Tiling and Rhythmic Canons: A CP Journey

Guillaume Derval[1], **Christophe Lecoutre**[2]

[1]University of Liège, Montefiore Institute, Smart Grids Lab, Liège, Belgium
[2]Univ. Artois & CNRS, CRIL, Lens, France
gderval@uliege.be, lecoutre@cril.fr

Abstract

The Aperiodic Tiling Complements Problem (ATCP) involves finding the full set of (normalized) aperiodic complements of a given pattern. This has become a classic problem in music theory, with some recent attempts to model it using Integer Linear Programming (ILP) and Boolean Satisfiability (SAT) frameworks. In this paper, we develop and compare different models of ATCP encoded with Constraint Programming (CP). The most effective approach admits two phases: a first one that allows us to merge (join) several subsets of linear constraints under the form of tables with large arity, and a second one that advantageously exploits the generated tables to discard periodic tiling complements. Our experimental results show that our approach significantly outperforms the state-of-the-art, solving every instance of a classical benchmark (standard Vuza rhythms for canons with periods set up to 900) in a time between 5 seconds and 2 minutes (except the largest instance being solved in 18 minutes).

1 Introduction

The art of composing music undoubtedly dates back a long way in human evolution. Interestingly, music has a close relationship with mathematics. While music is not (solely) the pursuit of mathematical perfection, composers generally seek (above all) solutions to problems of harmony, melody and rhythm. For Johann Sebastian Bach, the foundation of musical creation is the counterpoint. This is the science of superimposing two (or more) phrases, point-to-point, which complement each other in their behavior.

Since the advent of the digital world, modern composers have had access to tools that are, for the most part, beneficial (and, perhaps, occasionally detrimental). The first example of the use of computers for musical creation (actually, random processes) appears to date back to 1957, when Hiller and Isaacson introduced the Illiac Suite (a string quartet). Since then, numerous works have been carried out combining music and mathematics (and/or computational tasks), with many interesting questions and problems studied and disseminated in the literature.

Constraint Programming (CP) has been shown to be an appropriate technology for the computational modeling of music theories and composition (Pachet and Roy 2001; Anders and Miranda 2011; Pachet et al. 2015). This has led to the development of several systems (Zimmermann 2001; Truchet and Codognet 2004; Sandred 2010; Anders 2018).

Since Dan Tudor Vuza (1991-93) established the theoretical foundations of Tiling Rhythmic Canons, the computational aspects of this musical model have been investigated. In particular, the count of the number of possible rhythmic patterns that tile the time horizon by translation while avoiding periodicity has been examined. More specifically, the problem can be stated as follows: given an integer n denoting the time period, we want exactly one of the two possible voices (denoted A and B) to play at every time instant t in the integer time interval ranging from 0 to $n-1$. A tiling rhythmic canon is then defined as a pair of sets, A and B, such that $A \subseteq \mathbb{Z}_n$, $B \subseteq \mathbb{Z}_n$, and for every $t \in \mathbb{Z}_n$, there exists a unique pair, $(a, b) \in A \times B$, such that $a+b \bmod n = t$, where $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$. While it is easy to find tiling canons in which at least one voice is periodic (i.e., it repeats a shorter rhythm), the most computationally complex canons are those in which both voices are aperiodic.

The Aperiodic Tiling Complements Problem (ATCP) involves computing the full set of (normalized) aperiodic complements of a given set (rhythm or inner voice) A. The most recent approaches to solving the ATCP are based on different paradigms. The first of these is the Fill-Out procedure, which was introduced in (Kolountzakis and Matolcsi 2009): it relies on a heuristic that expands a complement of the first (inner) voice A step by step. However, a post-processing step is necessary to discard periodic solutions and symmetrical solutions. An Integer Linear Programming (ILP) model was proposed in (Auricchio, Ferrarini, and Lanzarotto 2023), based on the polynomial characterization of tiling canons. As some constraints are dynamically added when canons are found, no post-processing step is required. Finally, this ILP model was revisited and an original SAT encoding was proposed in (Auricchio et al. 2024). The SAT approach substantially outperforms the other approaches. Almost all instances in a representative benchmark, including various rhythms and periods, have been solved. With the most effective SAT model, combined with the right level of decomposition and parallelization, only 2 instances of this benchmark remained unsolved within 3 hours, and one instance took $2,226$ seconds.

In this paper, we show how all instances of this benchmark can be solved using a CP approach, each in less than 2 minutes (except for a single one in 18 minutes); for some instances, our approach is at least two orders of magnitude faster than the state-of-the-art.

2 Problem Description

The rhythmic canons problem can be seen as the problem of tiling integer intervals. Let n be a positive integer (denoting the period of a canon), $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ be the cyclic group of remainder[1] classes modulo n, and $\mathbb{Z}_n^+ = \mathbb{Z}_n \setminus \{0\}$. Subsets of $\mathbb{Z}_n$ are called **rhythms**. Given two rhythms, $A \subseteq \mathbb{Z}_n$, called the **inner voice**, and $B \subseteq \mathbb{Z}_n$, the **outer voice**, we want to ensure that they together tile $\mathbb{Z}_n$.

Definition 1. *Let $A, B \subseteq \mathbb{Z}_n$. A and B are said to **tile** $\mathbb{Z}_n$ if*

$$\exists!(a, b) \in A \times B : a + b = z \pmod{n} \quad \forall z \in \mathbb{Z}_n,$$

i.e. if every integer is the sum of a single pair of elements in A and B. Equivalently, A and B tile $\mathbb{Z}_n$ if

$$|A| \cdot |B| = |\mathbb{Z}_n| \qquad \textit{and} \qquad A \oplus B = \mathbb{Z}_n.$$

where $A \oplus B = \{a + b \mid a \in A, b \in B\}$. If A and B tile $\mathbb{Z}_n$, the tuple (A, B) is said to be a **tiling rhythmic canon** *with period n.*

Example 1. *For $n = 9$ and $A = \{0, 3, 6\}$, one can observe that $B = \{0, 1, 5\}$ is a tiling complement of A. This is because $A \oplus B = \{0, 3, 6\} \oplus \{0, 1, 5\} = \{0, 1, 5, 3, 4, 8, 6, 7, 11\%9\} = \{0, 1, 2, 3, 4, 5, 6, 7, 8\} = \mathbb{Z}_9$.*

Concerning notations, the size of a set (usually, denoted by a single upper-case letter) will be denoted by the same letter in a monospaced lower-case font. For example, the sizes of A and B will be denoted by $\mathtt{a}$ and $\mathtt{b}$.

In this article, we are interested in finding aperiodic rhythms. Periodic rhythms are less interesting because they are less structured and less computationally challenging, as discussed in (Kolountzakis and Matolcsi 2009). Aperiodicity is defined from the concept of translation. The notation $A + z$, where $A \subseteq \mathbb{Z}_n$ and z is an integer, corresponds to a translation (everything taken modulo n) defined as follows:

$$A + z = \{a + z \bmod n \mid a \in A\}.$$

Definition 2. *A rhythm $A \subseteq \mathbb{Z}_n$ is **periodic** modulo a value $z \in \mathbb{Z}_n^+$ iff $A + z = A$, i.e., if translated by z (different from 0), we find A. A rhythm $A \subseteq \mathbb{Z}_n$ is periodic if there exists at least one value $z \in \mathbb{Z}_n^+$ such that A is periodic modulo z, **aperiodic** otherwise.*

Definition 3. *A tiling rhythmic canon (A, B) is a Vuza canon if both A and B are aperiodic.*

As indicated in (Auricchio et al. 2024), checking whether or not a rhythm is periodic is equivalent to checking whether or not there exists a value d in the set of maximal divisors of n such that the rhythm is periodic modulo d; this set is denoted by D_n (or D when the context is unambiguous). For example, for $n = 144$, we have $D_n = \{48, 72\}$ and for $n = 420$, we have $D_n = \{60, 84, 140, 210\}$.

[1]Throughout the paper, all operations are performed modulo n.

Another important property is that the tiling property is invariant under translation. That is, for any integers z_a, z_b, if A and B tile $\mathbb{Z}_n$, then $A + z_a$ and $B + z_b$ tile $\mathbb{Z}_n$. This leads to equivalence classes. For a given inner voice A, an outer voice B is equivalent to (at most if periodic, exactly if aperiodic) n other distinct rhythms: its translations. In this paper, we use multiple equivalence classes, depending on the context. The first one is an equivalence class where the representative is defined as the lexicographically smallest element among all possible translations.

Definition 4. *Given $A \subseteq \mathbb{Z}_n$, $\widetilde{A}$ is the lexicographically smallest element of the translation-based equivalence class containing A: $\widetilde{A} = \min_{lex}\{A + k \mid k \in \mathbb{Z}_n\}$. If a set $A = \widetilde{A}$, A is said to be **l-normalized**.*

Here, it is assumed that sets are naturally ordered. For $n = 9$ and $A = \{0, 1, 5\}$, we have $A + 0 = \{0, 1, 5\}$, $A + 4 = \{0, 4, 5\}$, $A + 8 = \{0, 4, 8\}$, while all other translations $A + k$, for $k \in \mathbb{Z}_9 \setminus \{0, 4, 8\}$ do not contain 0 (and so, cannot be representatives). Hence $\widetilde{A} = A$. We also use this second equivalence class:

Definition 5. *Given $A \subseteq \mathbb{Z}_n$, $\check{A}$ is the translation of A such that its additive inverse $-\check{A} = \{n - a \bmod n \mid a \in \check{A}\}$ is the lexicographically largest element of the equivalence class containing $-A$: $-\check{A} = \max_{lex}\{-A + k \mid k \in \mathbb{Z}_n\}$. If a set $A = \check{A}$, A is said to be **r-normalized**.*

Note that 0 is always a member of a l- or r-normalized rhythm. We can now define the subject of our study: modeling and solving the Aperiodic Tiling Complements Problem.

Definition 6. *Given a period n and a rhythm $A \subseteq \mathbb{Z}_n$, the Aperiodic Tiling Complements Problem (ATCP) involves finding all normalized aperiodic complements B of A, i.e., all rhythms $B \subseteq \mathbb{Z}_n$ that are both aperiodic and normalized and such that A and B tile $\mathbb{Z}_n$.*

The term *normalized* either indicates l-normalized or r-normalized. Solving the ATCP by finding all l-normalized complements $\widetilde{B}$ is equivalent up to a translation to finding all r-normalized complements $\check{B}$, and vice versa.

3 The Journey

To solve the ATCP, we opted for constraint programming. This involved testing various ideas, which resulted in different models. Here, we describe this process, the *journey* to our final solution, explaining the strengths and limitations of each new step.

Constraint Programming. Constraint Programming (CP) is used to model and solve instances of the Constraint Satisfaction Problem (CSP), which are defined by constraint networks. A Constraint Network (CN) consists of a finite set of variables subject to a finite set of constraints. Each variable x can take a value from a finite set (of integers) $dom(x)$ called the domain of x. Each constraint c is specified by a relation that is defined over (the Cartesian product of the domains of) a subset of variables (called the *scope* of the constraint); the *arity* of a constraint c is the number of variables involved in c. A solution of a CN is

the assignment of a value to every variable such that all constraints are satisfied. In the literature, there exist many forms of constraints with specific semantics; they are called *global* constraints. However, CP solvers usually recognize the same kernel of popular constraints (around 20 global constraints, including their variants). In our context, we shall need five global constraints:

- The constraint `Sum` is certainly the most recurrent constraint. Its general form is $\sum_{i \in 1..r} c_i \times x_i \odot k$ where c_i is an integer, x_i a variable, $\odot$ a relational operator (for example, $\leq$), and k an integer (limit).
- The constraint `AllDifferent` is also frequently used, and well known (Régin 1994; van Hoeve 2001; Gent, Miguel, and Nightingale 2008). It ensures that the variables in a specified list must all take different values.
- The constraint `Lex` (Carlsson and Beldiceanu 2002; Frisch et al. 2002) ensures that the tuple formed by the values assigned to the variables of a first list is related to the tuple formed by the values assigned to the variables of a second list with respect to a lexicographic order operator, such as, e.g., $<_{lex}$. For example the constraint $\langle x_1, x_2, x_3 \rangle <_{lex} \langle y_1, y_2, y_3 \rangle$ is satisfied if x_1, x_2, y_1 are assigned to 0 and y_2 is assigned to 1 (whatever the values assigned to x_3 and y_3).
- The constraint `NValues` (Bessiere et al. 2006), ensures that the number of distinct values taken by the variables of a specified list respects a numerical condition. For example, $\texttt{NValues}(x_1, x_2, x_3, x_4) < 2$ is satisfied if x_1 and x_2 are assigned to 0 while x_3 and x_4 are assigned to 1.
- The constraint `Table` is defined by enumerating in a set the tuples of values that are allowed for a sequence of variables. Some algorithms for table constraints are STR2 (Lecoutre 2011) and CT (Demeulenaere et al. 2016; Verhaeghe, Lecoutre, and Schaus 2017).

Experimental Environment. At the time this paper was written, the state-of-the-art was the SAT approach described in (Auricchio et al. 2024). We reused their dataset, composed of 36 instances with a time horizon n ranging from 72 to 900. Some of these instances are classical ones by Vuza (1991-93), Fripertinger (2005), Amiot (2009), Kolountzakis and Matolcsi (2009). While introducing successive models, we compare our experimental results with those in (Auricchio et al. 2024) (denoted by SOTA, standing for state-of-the-art). In our approach, all models have been written with the Python library PyCSP^3 (Lecoutre and Szczepanski 2020), and all instances are solved with the Java constraint solver ACE (Lecoutre 2023). From the dataset, we excluded instance 29, which has trillions of solutions which are obviously not enumerable in our lifetime.

The machine is powered by an Intel Xeon Gold 6140 CPU @2.30GHz with 252 GB of RAM. Each solver is given a single thread.

Instances				Solving time (s)	
#	n	a	# sols	Model m1	SOTA
1	72	6	6	2.11	0.00
2	108	6	252	124.10	0.10
3	120	6	18	103.80	0.01
4	120	10	20	7.34	0.01
5	144	6	36	271.20	0.02
6	144	6	8640		11.88
7	144	12	6	11.50	0.00
8	144	12	60	29.00	0.02
9	168	6	54		0.04
10	168	14	42	30.70	0.02
11	180	6	2052		2.84
12	180	6	96		0.07
13	180	10	1800		1.39
14	180	15	120	75.10	0.05

Table 1: Solving times (in seconds) obtained with the first model and SOTA on the first 14 instances of our benchmark. Blank values indicate a timeout after 300 seconds.

3.1 First Model with Integer Domains

A straightforward way to start writing a model for the ATCP (Aperiodic Tiling Complements Problem) is to introduce an array x of $\texttt{b}$ integer variables that represent the set B that must be computed: $x_i \in X$ is the ith value of a normalized aperiodic tiling complement B of A (which is given as input); we have $dom(x_i) = \mathbb{Z}_n, \forall i \in 1..\texttt{b}$. A basic CP model is then obtained by:

- ensuring that B is the outer voice of a tiling rhythmic canon with inner voice A by posting a single constraint AllDifferent on all expressions of the form $a_i + x_j$ (for any i and for any j); see Equation (m1.1),
- ensuring that B is an aperiodic rhythm by posting a constraint NValues per maximal divisor d of n; see Equation (m1.2). If there are more than b distinct values, it means that B is not periodic with respect to the divisor d.

$$\texttt{AllDifferent}(a_i + x_j : i \in 1..\texttt{a} \wedge j \in 1..\texttt{b}) \quad \text{(m1.1)}$$

$$\texttt{NValues}(x_1, \ldots, x_\texttt{b}, x_1 + d, \ldots, x_\texttt{b} + d) > \texttt{b} \quad \forall d \in D \quad \text{(m1.2)}$$

$$x_i \in \mathbb{Z}_n \quad \forall i \in 1..\texttt{b} \quad \text{(m1.3)}$$

To ensure normalization (we are looking for l-normalized rhythms), we need to post some symmetry-breaking constraints. This is made possible by introducing first a two-dimensional array y of variables: $y_{k,i}$ is the ith value of the kth tiling complement equivalent to x under translation that contains 0; there are $\texttt{b}-1$ such translations. We have $dom(y_{k,i}) = \mathbb{Z}_n, \forall k \in 1..\texttt{b}-1, \forall i \in 1..\texttt{b}$. Symmetries are broken by: (i) ensuring values are strictly ordered in x (and, so, in B), see Equation (m1.4), (ii) computing the translations of B, see Equations (m1.5) and (m1.6), (iii) ensuring that B is lexicographically strictly less than any of its translations, see Equation (m1.7).

$$x_i < x_{i+1} \; \forall i \in 1..\texttt{b}-1 \quad \text{(m1.4)}$$

$$y_{k,1} = n - x_{k+1} \wedge y_{k,k+1} = 0 \; \forall k \in 1..\texttt{b}-1 \quad \text{(m1.5)}$$

$$y_{k,i} = x_i + y_{k,1} \; \forall k \in 1..\texttt{b}-1, \; \forall i \in 2..\texttt{b} \mid i \neq k+1 \quad \text{(m1.6)}$$

$$x \prec_{lex} \langle y_{k,k+1}, \ldots, y_{k,k+1+\texttt{b}} \rangle \; \forall k \in 1..\texttt{b}-1 \quad \text{(m1.7)}$$

$$y_{k,i} \in \mathbb{Z}_n \; \forall k \in 1..\texttt{b}-1, \forall i \in 1..\texttt{b} \quad \text{(m1.8)}$$

These constraints imply that $x_1 = 0$, which can then be added to the model as a redundant (symmetry-breaking) constraint.

The results obtained using this model for the easiest instances are shown in Table 1. Compared to the SOTA, it behaves quite poorly since, within 300 seconds, it can only solve 9 instances out of the 35 instances of the dataset. There are multiple factors that certainly explain these poor performances: (a) the d constraints NValues are costly to propagate, (b) the constraint AllDifferent works best when the variables nearly form a permutation; here the number of variables is by far smaller than the number of values (i.e., $\mathtt{b} \ll n$), (c) there are $\mathtt{b}^2$ variables, each with a domain of size n, for a total of $\mathtt{b}^2 \times n$ values that must be maintained by the solver.

3.2 Second Model with Binary Representations

In (Auricchio, Ferrarini, and Lanzarotto 2023; Auricchio et al. 2024), several ILP models and SAT encodings are proposed for the ATCP. The key element of these approaches is a one-dimensional array y of n variables 0/1 that represents the characteristic vector of set B: y_i is 1 iff $i \in B$; we have $dom(y_i) = \{0, 1\}, \forall i \in 0..n-1$. In this model, there are thus only $2n$ values to be maintained by the solver.

Interestingly, it is possible to build a CP model in a much more direct and easier way than that required for encoding tiling, aperiodicity and normalization constraints in ILP and SAT frameworks. Indeed, ensuring both aperiodicity and normalization is made possible by imposing a lexicographic order on (inversed) characteristic vectors. More specifically, this actually corresponds to impose that rev(y) is a Lyndon word, where a Lyndon word (Lyndon 1954) is a nonempty string that is strictly smaller in lexicographic order than all of its rotations (circular shifts) and rev() is a function that reverses vectors.

Definition 7. *A 01-Lyndon word of length n is a vector of length n containing only values 0 and 1, which is the unique minimum element in the lexicographical ordering in the multiset of all its rotations.*

For example, 011 is a Lyndon word because $011 <_{lex} 110$ and $011 <_{lex} 101$. Because a Lyndon word is unique in the multiset of its rotations, this implies that a Lyndon word differs from any of its nontrivial rotations, and is therefore aperiodic. Thus, there is a very interesting link between normalized aperiodic rhythms and Lyndon words.

Proposition 1. *A rhythm $B \subseteq \mathbb{Z}_n$ is both aperiodic and r-normalized iff rev(B^{01}) is a Lyndon word (of length n) where B^{01} is the representation of B as a 01 vector ($B^{01}_i = 1 \iff i \in B$).*

We omit the formal proof. For our model, ensuring that rev(y) is a Lyndon word also guarantees that y is the 01 vector of a (r-)normalized aperiodic rhythm. To implement this, we can post $n-1$ constraints Lex. Interestingly, we can reduce the arity (number of involved variables) of such constraints as we have the following property:

Proposition 2. *A word w is a Lyndon word if and only if it is lexicographically **strictly** smaller than any of its suffixes.*

Example 2. *Given a rhythm $B = \{0, 2, 3\}$ over $\mathbb{Z}_6$, its 01-representation is $B^{01} = [1, 0, 1, 1, 0, 0]$, and its reverse is rev$(B^{01}) = [0, 0, 1, 1, 0, 1]$. If we were to prove that rev(B^{01}) is a Lyndon word without using Proposition 2, we would need to check that:*

$$[0,0,1,1,0,1] <_{lex} [0,1,1,0,1,0]$$
$$[0,0,1,1,0,1] <_{lex} [1,1,0,1,0,0]$$
$$[0,0,1,1,0,1] <_{lex} [1,0,1,0,0,1]$$
$$[0,0,1,1,0,1] <_{lex} [0,1,0,0,1,1]$$
$$[0,0,1,1,0,1] <_{lex} [1,0,0,1,1,0]$$

With Proposition 2, it is sufficient to check that each prefix of rev(B^{01}) is strictly smaller than the suffix of the same size:

$$[0] <_{lex} [1]$$
$$[0,0] <_{lex} [0,1]$$
$$[0,0,1] <_{lex} [1,0,1]$$
$$[0,0,1,1] <_{lex} [1,1,0,1]$$
$$[0,0,1,1,0] <_{lex} [1,1,0,0,1]$$

which reduces the number of comparisons.

Moreover, we can derive a property concerning the minimum number of consecutive trailing zeros in any solution.

Theorem 3. *Let w be a 01-Lyndon word of size n containing $k\,(>0)$ occurrences of value 1. The first $\lceil \frac{n-k}{k} \rceil$ values in the vector (word) are equal to 0 and the last value of the vector is equal to 1.*

Proof. As w is a Lyndon word, the largest consecutive series (circularly) of 0 must be at the start of the word. Moreover, it must end with a 1. Note that $n-k$ is the number of 0 in w. In the worst case, two occurrences of 1 (there are k such pairs, circularly) must thus be separated by a $\frac{n-k}{k}$ series of 0s. In the case where $n-k$ is not divisible by k, one of these series of 0 must be longer than the others. □

Theorem 3 can be directly applied to our model. The array of variables y, when the reverse-Lyndon-word property is enforced, must start with a 1 (B contains 0) and must end with (at least) $\lceil \frac{n-\mathtt{b}}{\mathtt{b}} \rceil$ 0s (B does not contain $n-1, n-2, \ldots, n-\lceil \frac{n-\mathtt{b}}{\mathtt{b}} \rceil$). This leads to the following constraints:

$$y_0 = 1 \tag{m2.1}$$
$$y_i = 0 \;\; \forall i \in n - \lceil \frac{n-\mathtt{b}}{\mathtt{b}} \rceil ..n-1 \tag{m2.2}$$
$$\sum_{i \in 0..n-1} y_i = \mathtt{b} \tag{m2.3}$$
$$\langle y_{n-1}, \ldots, y_{n-i} \rangle <_{lex} \langle y_{i-1}, \ldots, y_0 \rangle \; \forall i \in 1..n-1 \tag{m2.4}$$
$$y_i \in \{0,1\} \;\; \forall i \in 0..n-1 \tag{m2.5}$$

Tiling remains to be enforced in our model. This can be done with circulant matrices. Given a period n and a rhythm $A \subseteq \mathbb{Z}_n$, let $A^{01} = \langle a_0, \ldots, a_{n-1} \rangle$ be the characteristic vector of A, where $a_i = 1$ if $i \in A$ and $a_i = 0$ if $i \notin A$. As shown in (Auricchio et al. 2024), from A^{01}, we can define the circulant matrix $M^A \in \{0,1\}^{n \times n}$ of rhythm A, where each column of M^A is the circular shift of the first column corresponding to A^{01} transposed. Thus, the matrix M^A is equal to:

$$\begin{bmatrix} a_0 & a_{n-1} & a_{n-2} & \ldots & a_1 \\ a_1 & a_0 & a_{n-1} & \ldots & a_2 \\ a_2 & a_1 & a_0 & \ldots & a_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_0 \end{bmatrix}$$

Instances				Solving time (s)		
#	n	a	# sols	Model m1	Model m2	SOTA
1	72	6	6	2.11	0.74	0.00
2	108	6	252	124.10	1.98	0.10
3	120	6	18	103.80	1.91	0.01
4	120	10	20	7.34	1.72	0.01
5	144	6	36	271.20	1.90	0.02
6	144	6	8640		5.43	11.88
7	144	12	6	11.50	1.18	0.00
8	144	12	60	29.00	1.73	0.02
9	168	6	54		1.43	0.04
10	168	14	42	30.70	2.61	0.02
11	180	6	2052		2.13	2.84
12	180	6	96		2.03	0.07
13	180	10	1800		2.72	1.39
14	180	15	120	75.10	4.41	0.05
15	180	6	281232		31.30	
16	420	10	720		4.34	2.39
17	420	14	672		5.86	1.73
18	420	15	3120		85.90	8.28
19	420	21	1008		11.50	1.77
20	420	6	864		7.31	5.32
21	420	14	6720			14.23
22	420	15	33480			114.91
23	420	35	840		10.90	0.85
24	420	6	1872		28.20	12.20
25	420	21	10080			18.13
26	420	10	22320			97.23
27	420	35	1120		21.60	1.34
28	420	14	40572		34.20	
30	900	75	15600			47.92
32	900	30	15840		252.60	119.02

Table 2: Solving times (in seconds) obtained with the first model, second model and SOTA on the instances (of our benchmark) solved by at least one of them before timeout. Blank values indicate a timeout after 300 seconds.

This circulant matrix M^A can be used to impose the tiling conditions:

$$\sum_{i \in 0..n-1} M^A_{ij} \times y_i = 1 \qquad j \in 0..n-1 \qquad \text{(m2.6)}$$

The full model, using equations (m2.1) to (m2.6), is named "Model m2" in the experiments. We show in Table 2 the results of running the model on some instances. It is more competitive than the previous model compared to the state of the art, and is even able to solve the instances 15 and 28 while the current SOTA timeouts, but the model is still slower for most instances.

3.3 Divide & Conquer Model

Although the experimental results obtained from the second CP model represent a significant improvement on those from the first model, they are still mixed compared to the SAT approach. To enhance performance, we propose incorporating a pre-solving step to break down the problem. This approach is presented in this section.

Consider the set C of tiling constraints from Equation (m2.6):

$$C = \left\{ \sum_{i \in 0..n-1} M^A_{ij} \times y_i = 1 : j \in 0..n-1 \right\}$$

The scopes (involved variables) of the constraints typically overlap.

Example 3. *For $n = 9$ and $A = \{0, 3, 6\}$, we have $A^{01} = \langle 1, 0, 0, 1, 0, 0, 1, 0, 0 \rangle$, and the following constraints in C:*

$$y_0 + y_3 + y_6 = 1\ (c_0) \qquad y_1 + y_4 + y_7 = 1\ (c_1)$$
$$y_2 + y_5 + y_8 = 1\ (c_2) \qquad y_3 + y_6 + y_0 = 1\ (c_3)$$
$$y_4 + y_7 + y_1 = 1\ (c_4) \qquad y_5 + y_8 + y_2 = 1\ (c_5)$$
$$y_6 + y_0 + y_3 = 1\ (c_6) \qquad y_7 + y_1 + y_4 = 1\ (c_7)$$
$$y_8 + y_2 + y_5 = 1\ (c_8)$$

Looking at the scope of these linear constraints, one can observe that many of them share variables. For this basic example, the overlapping of variables is even very particular, as, for example, the first (c_0), fourth (c_3), and seventh (c_6) constraints are strictly equivalent.

From this observation, if c_j denotes the $j-1$th linear constraint in C obtained from M^A (we know that we have n such constraints), and g is a divisor of n, one can define a partition of C as follows:

$$C^g_k = \{c_j \in C \mid j \bmod g = k\}, \forall k \in 0..g-1$$

These g proper subsets of C are called (tiling) *constraint parts* hereafter. We refer to g as the gap in C between two successive constraints that are selected in the same part. Each constraint part contains n/g constraints. We define as $scp(C^g_k)$ the (ordered) sequence of variables involved in C^g_k (i.e., the ordered union of constraint scopes). These constraint parts (disjoint subsets of C) have interesting properties:

- each subset of constraints involves constraints whose scopes overlap potentially quite greatly, some constraints even being redundant;
- depending on the partition, little to no variables are shared between the scopes of the subsets;
- all subsets are equivalent up to a translation: one can pass from a subset to any other one by a simple bijection applied to variable indices;
- the number of solutions of each subset of constraints is limited in practice (at least on classical benchmarks, as we shall see in our experiments).

Example 4. *For Example 3, if $g = 3$ (a divisor of $n = 9$), we obtain $C^3_0 = \{c_0, c_3, c_6\}$, $C^3_1 = \{c_1, c_4, c_7\}$, and $C^3_2 = \{c_2, c_5, c_8\}$. As some linear constraints are identical, we actually have only one constraint in each part: $C^3_0 = \{y_0 + y_3 + y_6 = 1\}$, $C^3_1 = \{y_1 + y_4 + y_7 = 1\}$, and $C^3_2 = \{y_2 + y_5 + y_8 = 1\}$. Here, note that all constraint parts are independent (i.e., do not share variables) and equivalent (for example, one can pass from C^3_0 to C^3_1 simply by increasing the indices of variables involved in C^3_0).*

From these observations, we propose a two phase modeling/solving process. The first phase involves computing all solutions of the constraint parts (it suffices to compute the solutions of the first constraint part, as parts are all equivalent up to the indices of the involved variables). The second phase is the reformulation of the second CP model, developed in Section 3.2, by replacing the n tiling constraints with n/g table constraints corresponding to the solutions of the constraint parts. We first discuss the choice of g.

Choice of the Gap g Some choices of g may drastically decrease the number of constraints and the arity of constraint parts, and thus both the pre-solving time and the number of solutions.

While the relationship between A and the optimal choice of g is complex, we provide two theorems showing that some choices of g reduce the overlapping existing between the scopes of constraint parts and that some ATCP instances can even be optimally divided into constraint parts having non-overlapping scopes. In the theorems below, we write $A \bmod g = \{a \bmod g \mid a \in A\}$.

Theorem 4. *If g is a divisor of n such that $g > 1$ and $|A \bmod g| < g$ then the scopes of the constraint parts $C_0^g, C_1^g, ...$ do not contain all the variables:*

$$|scp(C_k^g)| < n \qquad \forall k \in 0..g-1$$

Theorem 5. *If g is a divisor of n such that $g > 1$ and $|A \bmod g| = 1$, then the scopes of the constraint parts $C_0^g, C_1^g, ...$ are disjoint.*

We omit the proofs due to a lack of space; they rely on standard group theory results. The proofs are available in the supplementary material of this paper.

In the experiments, we systematically choose for each instance the value g that minimizes $|\text{scp}(C_0^g)|/|C_0^g|$, with $|\text{scp}(C_0^g)| \neq n$. The smallest g is kept in case of ties. This acts as a proxy to minimize both the number of tuples in T and the number of table constraints.

First Phase In the first phase of the third CP approach, we are looking for the solutions T_k of each tiling constraint part C_k^g (and so, only variables involved in part k are considered, i.e., $\text{scp}(C_k^g)$). As mentioned previously, the constraint parts C_k^g are all equivalent up to a translation of the variable indices: the set of solutions (tuples) of one part is valid for the other parts. We thus drop the subscript and simply write the set of solutions as T.

In order to further restrict the number of enumerated solutions (in T), we can already exclude some of them that will never participate in a solution of the complete model (second phase). For this, we can use the trailing-0s property derived from Theorem 3: a solution to the complete problem (model), when using the reverse-Lyndon-word constraint, must end with at least $z = \lceil \frac{n-\mathtt{b}}{\mathtt{b}} \rceil$ $0s$. However, because our objective is to precompute a single (common) set T of solutions for all constraint parts, we need to be careful. We cannot systematically enforce $y_i = 0$ if $i \geq n - z$. Indeed, we must take into account the translations. A valid condition (that can be used when computing T) becomes $y_i = 0$ if $n - z \leq i < n - g$.

Example 5. *Let us assume that $n = 9$, $g = 3$, and $z = 2$. Clearly, y_8 must be equal to 0 in a solution to the complete problem (model). The variable y_8 in C_0^3 corresponds to the variable y_2 in C_2^3, due to the translation; If enforcing $y_8 = 0$ is correct, enforcing $y_2 = 0$ would be an error. The last $g - 1$ variables must then be excluded from the trailing-0s constraints when computing T.*

Instances				Solving time (s)		
#	n	a	# sols	m2	m3	SOTA
18	420	15	3120	85.90	60.33	8.28
23	420	35	840	10.90	5.86	0.85
27	420	35	1120	21.60	8.39	1.34
32	900	30	15840	252.60	111.20	119.02

Table 3: Solving times (in seconds) obtained with the second model, third model and SOTA on a selection of instances where the behaviors of the second and third models differ.

Second Phase After having chosen a value of g, we can reformulate the set C of n linear constraints used to impose tiling conditions as a set C' of n/g table constraints defined as follows:

$$C' = \{scp(C_k^g) \in T : k \in 0..n/g - 1\}$$

where T is the set of solutions computed (in the first phase) for any constraint part C_k^g.

Example 6. *For Example 3, we have only 3 solutions for each constraint part, and so we have $T = \{(0,0,1),(0,1,0),(1,0,0)\}$, and C' composed of the three following table constraints:*

$$\langle y_0, y_3, y_6 \rangle \in T \quad \langle y_1, y_4, y_7 \rangle \in T \quad \langle y_2, y_5, y_8 \rangle \in T$$

The "Divide and Conquer" model m3 corresponds to the "Binary model" m2 of Section 3.2 by replacing Equation (m2.6) with the following Equation:

$$scp(C_k^g) \in T, \forall k \in 0..n/g - 1 \qquad \text{(m3.1)}$$

The model m3 has very similar running time compared to the previous model m2, except in some specific instances shown in Table 3. The model's runtime suffers from multiple startup costs (as the two phases must be started independently). The improvement obtained via the separation of the problem in tables is not sufficient to overcome these costs on small problems.

3.4 Aperiodicity Useful for Divide & Conquer

The previous model does not exploit the information made available after computing T; aperiodicity is ignored during Phase 1. In this section, we explore how to precompute periodicity information for each constraint part C_k^g in Phase 1, in order to exploit this information in Phase 2. More precisely, we extend the concept of periodicity to the tuples that are present in T.

Definition 8. *A tuple $t = \langle t_{i_0}, t_{i_1}, \ldots, t_{i_{m-1}} \rangle \in T$ associated with a scope $scp(C_k^g) = \langle y_{i_0}, y_{i_1}, \ldots, y_{i_{m-1}} \rangle$ is periodic with modulo d iff*

$$y_{(i_j+d) \bmod n} \in scp(C_k^g) \wedge t_{(i_j+d) \bmod n} = t_{i_j} \quad \forall j \in 0..m-1$$

Theorem 6. *If all tuples associated with vector y are periodic modulo d, then y is periodic modulo d.*

Proof. To each constraint part C_k^g, there is a tuple $t_k = \langle t_{k,(i_0+k) \bmod n}, \ldots, t_{k,(i_{m-1}+k) \bmod n} \rangle \in T$ associated with the vector y and the scope

$\text{scp}(C_k^g) = \langle y_{(i_0+k) \bmod n}, \ldots, y_{(i_{m-1}+k) \bmod n} \rangle$ such that $y_{(i_j+k) \bmod n} = t_{k,(i_j+k) \bmod n} \forall j \in 0..m-1, k \in 0..g-1$. As $\bigcup_k \text{scp}(C_k^g) = y$ (all the variables are in the union of the scopes), the condition from Definition 8 is met by all variables - and having $y_{(j+d) \bmod n} = y_j \quad \forall j \in 0..n-1$ is the definition or periodicity modulo d. □

The interesting result is actually the contrapose of the last theorem, as we are looking for an aperiodic rhythm: a rhythm encoded in the vector y is not periodic with modulo d if at least one of the associated tuple is not periodic of modulo d. Recall that it is sufficient to test aperiodicity only for maximal divisor (Auricchio et al. 2024) of n (contained in the set D). We use this result in order to further cut domains' sizes during solving.

We introduce a two-dimensional array p of $n/g \times |D|$ variables: $p_{k,l}$ is 1 if the (sub-)tuple chosen from the kth table constraint (in C') is periodic with respect to the lth divisor in D. This information is precomputed at the end of phase 1 for each tuple of T. More specifically, we transform the table T into a table T^{ext} where each tuple $t \in T$ of length n/g is extended to become a tuple t^{ext} of length $n/g + |D|$ by adding a prefix of length $|D|$ indicating for each divisor $d \in D$ if the (sub-)tuple t is periodic modulo d. We have $T^{ext} = \{(\rho(t, d_0), \rho(t, d_1), \ldots, \rho(t, d_{|D|-1})) \oplus t \mid t \in T\}$, where $D = \{d_0, d_1, \ldots, d_{|D|-1}\}$, $\oplus$ denotes the concatenation of tuples, and $\rho(t, d_i)$ returns 1 if the tuple τ is periodic of period d, 0 otherwise.

We can now use the results above and enforce that not all selected tuples (there are g selections to do, one per table constraint) share the same periodicity:

$$\sum_{k \in 0..\mathtt{g}-1} p_{k,l} < g \qquad \forall l \in 0..\mathtt{d}-1$$

The extended "Divide and Conquer" model corresponds to the "Binary model" of Section (3.2) by replacing Equation (m2.6) with the following Equations:

$$\{p_{k,0}, \ldots, p_{k,\mathtt{d}-1}\} \oplus \text{scp}(C_k^g) \in T^{\text{ext}} \qquad \forall k \in 0..n/g-1 \quad \text{(m4.1)}$$

$$\sum_{k \in 0..\mathtt{g}-1} p_{k,l} < n/g \qquad \forall l \in 0..|D|-1 \quad \text{(m4.2)}$$

$$p_{k,l} \in \{0,1\} \qquad \forall k \in 0..n/g-1, \ \forall l \in 0..|D|-1 \quad \text{(m4.3)}$$

Table 4 shows the runtime of the fourth model. While the Java & Python startup cost still dominates the solving time for smaller/simpler problems, the model is able to solve all instances in less than 180 seconds, except instance 33 in $1,093$ seconds, solving 7 instances more than the current SOTA of (Auricchio et al. 2024). A large part of the solving time is actually I/O between Python and Java and data processing rather than solving; an end-to-end implementation in a low-level language would be even faster.

To summarize, model m4 outperforms the SOTA, using a combination of three ideas presented in this paper: (a) symmetry-breaking via Lyndon word enforcement, (b) divide and conquer via constraint tabling, and (c) computing periodicity information for the tables.

4 Supplementary Material

Due to length limitations, Table 1, 2, 3, 4 do not present the whole results for each model. They are available in the supplementary material in Table A.1, along with the number of tuples in T and the chosen g for each instance. Table A.2 displays the arity and number of constraints obtained for various values of g on the instances. Table A.3 presents more detailed information on the runtime of each phase of model m4. Proofs of Theorems 4 and 5 are also available in the Supplementary Material. The code to reproduce the experiment is available. It uses Snakemake (Möder et al. 2021) to allow reproducibility.

Instances				Solving time (s)		
#	n	a	# sols	m3	m4	SOTA
15	180	6	281232	25.93	23.23	
18	420	15	3120	60.33	6.47	8.28
19	420	21	1008	7.59	5.12	1.77
20	420	6	864	4.31	3.09	5.32
21	420	14	6720		8.82	14.23
22	420	15	33480		21.56	114.91
23	420	35	840	5.86	4.14	0.85
24	420	6	1872	27.23	3.97	12.20
25	420	21	10080		11.32	18.13
26	420	10	22320		17.80	97.23
27	420	35	1120	8.39	3.81	1.34
28	420	14	40572	35.70	24.50	
30	900	75	15600		55.51	47.92
31	900	15	235200		128.67	
32	900	30	15840	111.20	32.60	119.02
33	900	10	1302000		1093.80	
34	900	45	118080		152.20	
35	900	15	123840		119.25	
36	900	30	62160		82.16	

Table 4: Solving times (in seconds) obtained with the third model, fourth model and SOTA on a selection of large instances. Blank values indicate a timeout after 300 seconds. A specific timeout of $2,000$ seconds was used for the most difficult instance (33).

5 Conclusion

Constraint Programming remains a tool of choice for declaratively modeling (and solving) many combinatorial problems. In particular, the universal nature of table constraints is, in some cases, a means of breaking a practical resolution deadlock. This is what we demonstrate in this article for the ATCP problem, where a two-step approach allowing table constraints to be generated in preprocessing, along with the usage of some problem properties, such as Lyndon words' properties and aperiodicity constraints, has enabled us to outperform state-of-the-art approaches to this problem.

Acknowledgments

This work has benefited from the support of the National Research Agency under France 2030, MAIA Project (ANR-22-EXES-0009).

References

Amiot, E. 2009. New perspectives on rhythmic canons and the spectral conjecture. *Journal of Mathematics and Music*, 3(2): 71–84.

Anders, T. 2018. Compositions created with constraint programming. *Chapter 10 of the Oxford Handbook of Algorithmic Music*.

Anders, T.; and Miranda, E. R. 2011. Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys*, 43(4): 1–38.

Auricchio, G.; Ferrarini, L.; Gualandi, S.; Lanzarotto, G.; and Pernazza, L. 2024. Computing aperiodic tiling rhythmic canons via SAT models. *Constraints*, 29(3-4): 215–233.

Auricchio, G.; Ferrarini, L.; and Lanzarotto, G. 2023. An integer linear programming model for tilings. *Journal of Mathematics and Music*, 17(3): 514–530.

Bessiere, C.; Hebrard, E.; Hnich, B.; Kiziltan, Z.; and Walsh, T. 2006. Filtering Algorithms for the NValue Constraint. *Constraints*, 11(4): 271–293.

Carlsson, M.; and Beldiceanu, N. 2002. Revisiting the lexicographic ordering constraint. Technical Report T2002-17, Swedish Institute of Computer Science.

Demeulenaere, J.; Hartert, R.; Lecoutre, C.; Perez, G.; Perron, L.; Régin, J.-C.; and Schaus, P. 2016. Compact-Table: efficiently Filtering Table Constraints with Reversible Sparse Bit-Sets. In *Proceedings of CP'16*, 207–223.

Fripertinger, H. 2005. Remarks on rhythmical canons. In *Proceedings of Colloquium on mathematical music theory*, 73–90.

Frisch, A.; Hnich, B.; Kiziltan, Z.; Miguel, I.; and Walsh, T. 2002. Global constraints for lexicographic orderings. In *Proceedings of CP'02*, 93–108.

Gent, I.; Miguel, I.; and Nightingale, P. 2008. Generalised arc consistency for the AllDifferent constraint: An empirical survey. *Artificial Intelligence*, 172(18): 1973–2000.

Hiller, L. A.; and Isaacson, L. M. 1957. Musical composition with a high speed digital computer. *Audio Engineering Society Convention 9*.

Kolountzakis, M. N.; and Matolcsi, M. 2009. lgorithms for translational tiling Supplementary sets and regular complementary unending canons (four parts). *Journal of Mathematics and Music*, 3(2): 85–97.

Lecoutre, C. 2011. STR2: Optimized Simple Tabular Reduction for Table Constraints. *Constraints*, 16(4): 341–371.

Lecoutre, C. 2023. ACE, a generic constraint solver. Technical Report arXiv:2302.05405, CoRR. Https://arxiv.org/abs/2302.05405.

Lecoutre, C.; and Szczepanski, N. 2020. PyCSP^3: Modeling Combinatorial Constrained Problems in Python. Technical Report arXiv:2009.00326, CoRR. Https://arxiv.org/abs/2009.00326.

Lyndon, R. C. 1954. On Burnside's problem. *Trans. Amer. Math. Soc.*, 77: 202–215.

Möder, F.; Jablonski, K.; Letcher, B.; Hall, M.; Tomkins-Tinch, C.; Sochat, V.; Forster, J.; Lee, S.; Twardziok, S.; Kanitz, A.; Wilm, A.; Holtgrewe, M.; Rahmann, S.; Nahnsen, S.; and Köter, J. 2021. Sustainable data analysis with Snakemake [version 1; peer review: 1 approved, 1 approved with reservations]. *F1000Research*, 10(33).

Pachet, F.; and Roy, P. 2001. Musical harmonization with constraints: A survey. *Constraints*, 6(1): 7–19.

Pachet, F.; Roy, P.; Papadopoulos, A.; and Sakellariou, J. 2015. Generating 1/f Noise Sequences as Constraint Satisfaction: The Voss Constraint. In *Proceedings of IJCAI'15*, 2482–2488.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of AAAI'94*, 362–367.

Sandred, O. 2010. PWMC, a Constraint-Solving System for Generating Music Scores. *Computer Music Journal*, 34(2): 8–24.

Truchet, C.; and Codognet, P. 2004. Musical constraint satisfaction problems solved with adaptive search. *Soft Computing*, 8(9): 633–640.

van Hoeve, W. 2001. The Alldifferent Constraint: a Survey. In *Proceedings of the Sixth Annual Workshop of the ERCIM Working Group on Constraints*.

Verhaeghe, H.; Lecoutre, C.; and Schaus, P. 2017. Extending Compact-Table to Negative and Short Tables. In *Proceedings of AAAI'17*, 3951–3957.

Vuza, D. 1991-93. Supplementary sets and regular complementary unending canons (four parts). *Perspectives of New Music*.

Zimmermann, D. 2001. Modelling musical structures. *Constraints*, 6(1): 53–83.