# Post-doc report 2015 - 2016 Tanaka Tsubasa (IRCAM)

Tsubasa Tanaka

Post-doc report 2015 - 2016

Tanaka Tsubasa
IRCAM
UMR 9912 STMS (IRCAM, CNRS, UPMC)
Equipe Représentations Musicale (Resp. G. Assayag)

My fundamental question in this postdoc research project is how to generate music that is reasonable and emotional. For this purpose, I focus on investigating how the global structure of an entire musical piece is constructed from basic units (vocabularies) that can be easily recognized, such as melodic motifs, rhythmic patterns, chord patterns, etc.

I first investigated the style of baroque polyphonic music, in which melodic motifs are important as the basic vocabulary. In polyphonic music, melodic motifs such as subject, response, countersubject, and other melodic figures are frequently repeated and the entire piece can be seen as a sort of puzzle that consists of such melodic fragments. To analyze the configuration of motifs in a musical piece and to reveal the structure of the piece, I developed a computational analysis that decomposes a polyphonic piece into a small number of classes of motif. Here, I adopt the class of motif instead of mere motif based on a contrapuntal viewpoint. In counterpoint, which is a theory of creating polyphonic music, motifs are seen as identical under certain types of transformations, i.e., inversion, retrograde, argumentation, diminution, transpositions. These transformations form a mathematical group. Using this transformation group, I can define equivalence classes of motif derived by the relation given by this group (this is to say that the motif after a transformation by this group and the motif before the transformation are related, and therefore belong to the same motif class).

Compared to a usual analysis by human, this computational method would allow us to accomplish more detailed analysis (melodic segmentation) detecting a set of motif classes that can reconstruct the entire structure. However, there are many possible solutions. To determine the best solution, I posed a constraint that fixes the number

of motif classes and I defined an objective function to be minimized as the number of motifs that appear in the segmentation. Minimizing this objective function is equivalent to maximizing the average length of the motifs. This means the optimal solution would provide a set of motif classes that cover the piece the most "efficiently." After giving this formulation, I found that this is a variation of the set-partitioning problem, which is well-known in the field of operations research, and I could implement this problem by integer programming, which is a highly developed tool to deal with computationally complex combinatorial problems.

To test this method, I analyzed J.S. Bach's Invention No.1. Thanks to the application of integer programming, I succeeded to obtain an optimal solution. In the resulting motifs, there were those that cover diverse characteristics such as the theme, countersubject, zigzag pattern, octave leap, irregular rhythmic pattern, etc. The distribution of these motifs within the piece gives us a lot of useful information to clarify the structure of the piece and the roles of each motif. For example, we could easily see that this piece consists of three sections, each consists of two parts, an exposition of the theme and a 1.5-measure ending part without an occurrence of the theme. Also, the irregular rhythmic pattern only appears directly before the 1.5-measure ending parts. This is as if this motif has a role for announcing the end of the sections. Although the zigzag pattern and stepwise pattern, which cause the sense of direction, only appear in ascending forms in the first two sections, these appear in descending forms in the last section. We can consider that the ascending forms of these motifs create the sense of expectation that the piece will continue further and that the descending forms create the sense of expectation that this piece will end soon. That implies, in the composition based on melodic motifs, the locations of motif and the types of transformation can have roles to control the perception of the listener. In this way, this analysis reveals the roles and the psychological/emotional meanings of the motifs and helps us to understand the global structure of the piece. Moreover, using the result of this analysis, I developed a method to visualize the relationship between acquired motifs using Hasse diagram. By seeing maximal elements of this diagram, we can clearly understand which motifs can derive the whole piece and the role of these motifs. For example, we can reduce J.S. Bach's Invention No.1 to only four small motifs (I call these four

motifs subject, countersubject, zigzag, and pre-cadence signal). Thus we can detect the basic elements of the piece. This analysis method was presented at International Congress of Mathematics and Music (ICMM). Concerning possible applications, this analysis would provide useful information to construct an algorithm to generate musical structure or to assist music performers to interpret the pieces.

Based on the same method (integer programming) and the same basic idea that musical pieces consist of localized musical vocabularies such as chords, rhythmic patterns, and melodic patterns, I formulated a method of generating global musical structures from a given set of localized musical patterns. This method allows us to generate a sequence of patterns that have a hierarchy and some global characteristics, as well as localized transition rules between the patterns. This formulation consists of only linear equations and linear inequalities and that allows us to apply integer programming. However, there is a limitation that only short sequences can be generated currently because finding a solution takes a lot of calculation time. So, section-wise generation is a realistic compromise for the moment. This method was presented at the conference Mathematics and Computation in Music (MCM2015).

Although the above methods were intended to treat classical music, I found, through some discussions with researchers whom I met at MCM2015, that similar method can be applied to the global structure of the pieces of Milton Babbitt, an American composer of 20th and 21st centuries who is famous for his mathematical ways of composition. In his certain compositions, he created an underlying structure called an all-partition allay, and then he realized the surface of the piece based on it. For Babbitt, all-partition allay is an extension of the maximal diversity principle of dodecaphonic music and it is a global structure that is formed by a covering of an given pitch class matrix by small sub-regions of the matrix whose shapes are determined by the type of integer partitions of 12 (an integer partition is a division of an integer such as 12 into a number of non-negative integers, e.g. 6+6+0+0 = 12 or 4+2+5+1=12.). Every type of integer partition must appear in an all-partition array. The problem of generating an all-partition array is a difficult

combinatorial problem and the method to solve it had not been revealed yet (although Babbitt and his student had a few solutions without using a computer and used them in composition.). To solve this problem by computer, I tried to formulate it by integer programming. In this formulation, I introduced a concept of duplication of the matrix elements between contiguous integer partition regions. Thanks to this, I found that this problem can be formalized as a variation of the set covering problem, which is well-known in the field of integer programming and is similar to the set-partitioning problem. Although I could formalize this problem by integer programming, I could not find a solution at that time because of the largeness of the problem. I presented only the formulation of this problem at the conference of International Society of Music Information Retrieval (ISMIR2016).

After this formulation, I reformulated the same problem by constraint programming, which is also effective to solve complex constraint satisfaction problem, and developed a new algorithm by integrating a statistic method and a divide-and-conquer method. As a result, I succeeded to find a solution for a four-row matrix, which is the smallest matrix that Babbitt used. This solution seems to be the first solution found by a computer. This result would be an important step toward automation of Babbitt's way of composition. This was presented at International Conference on Principles and Practice of Constraint Programming (CP2016). This was an important opportunity because I could introduce this difficult problem to the community of constraint programming research.

## Publication

[1] Tanaka, T., Fujii K., « Melodic Pattern Segmentation of Polyphonic Music as a Set Partitioning Problem », International Congress of Mathematics and Music (ICMM), Puerto Vallarta, November 26-29, 2014 (to be published by Springer).

[2] Tanaka, T., Fujii K., « Describing Global Musical Structures by Integer Programming on Musical Patterns », International Conference of Mathematics and Computation in Music (MCM) 2015, London, June 22-25, 2015. Lecture Notes in Computer Science, Vol. 9110. Springer, Berlin.

[3] Tanaka, T., Bemman, B. Meredith, D., « Integer Programming Formulation of the Problem of Generating Milton Babbitt's All-Partition Arrays », International Society for Music Information Retrieval Conference (ISMIR), New York, August 7-11, 2016.

[4] Tanaka, T., Bemman, B. Meredith, D., « Constraint Programming Approach to the Problem of Generating Milton Babbitt's All-Partition Arrays », International Conference on Principles and Practice of Constraint Programming (CP) 2016, Toulouse, September 5-9, 2016, Lecture Notes in Computer Science, Springer.

# Melodic Pattern Segmentation of Polyphonic Music as a Set Partitioning Problem

Tsubasa Tanaka and Koichi Fujii

**Abstract** In polyphonic music, melodic patterns (motifs) are frequently imitated or repeated, and transformed versions of motifs such as inversion, retrograde, augmentations, diminutions often appear. Assuming that economical efficiency of reusing motifs is a fundamental principle of polyphonic music, we propose a new method of analyzing a polyphonic piece that economically divides it into a small number of types of motif. To realize this, we take an integer programming-based approach and formalize this problem as a set partitioning problem, a well-known optimization problem. This analysis is helpful for understanding the roles of motifs and the global structure of a polyphonic piece.

## 1 Motif Division

In polyphonic music like fugue-style pieces or J.S. Bach's *Inventions and Sinfonias*, melodic patterns (motifs) are frequently imitated or repeated. Although some motifs are easy to find, others are not. This is because they often appear implicitly and/or appear in the transformed versions such as inversion, retrograde, augmentations, diminutions. Therefore, motif analysis is useful to understand how polyphonic music is composed.

Simply speaking, we can consider the motifs that appear in a musical piece to be economical if the number of types of motif is small, the numbers of repetitions are large, and the lengths of the motifs are long. Assuming that this economical efficiency of motifs is a fundamental principle of polyphonic music, we propose a new method of analyzing a polyphonic piece that efficiently divides it into a small number of types of motif. Using this division, the whole piece is reconstructed with

Tsubasa Tanaka
IRCAM and Tokyo University of the Arts, e-mail: tsubasa.tanaka@ircam.fr

Koichi Fujii
NTT DATA Mathematical Systems Inc. e-mail: fujii@msi.co.jp

a small number of types of motif like the puzzle game *Tetris*[1] (In tetris, certain domains are divided with only seven types of piece). We call such a segmentation a *motif division*.

If a motif division is accomplished, it provide us a simple and higher-level representation whose atom is a motif, not a note, and it will be helpful to clarify the structures of polyphonic music. The representation may provide knowledge about how frequent and where each motif is used, the relationships between motifs such as causality and co-occurrence, which transformations are used, how the musical form is constructed by motifs, and how the long-term musical expectations are formed. This analysis may be useful for applications such as systems of music analysis, performance, and composition.

Studies about finding boundaries of melodic phrases are often based on human cognition. For example, [2] is based on grouping principles of gestalt psychology, and [3] is based on a short-term memory model. While these studies deal with relatively short range of perception and require small amounts of computational time, we focus on global configuration of motifs on the level of compositional planning. This requires us to solve an optimization problem that is hard to solve. To deal with this difficulty, we take an *integer programming*-based approach [4] and show that this problem can be formalize as a *set partitioning problem* [5]. This problem can be solved by integer programming solvers that use efficient algorithms such as the branch and bound method.

## 2 Transformation Group and Equivalence Classes of Motif

In this section, we introduce *equivalence classes* of motif derived from a group of motif transformations as the criterion of identicalness of motifs. These equivalence classes are used to formulate the motif division in Sect. 3.

Firstly, a motif is defined as an ordered correction of notes $[N_1, N_2, \cdots, N_k]$ ($k > 0$), where $N_i$ is the information for the $i$th note, comprising the combination of the pitch $p_i$, start position $s_i$, and end position $e_i$ ($N_i = (p_i, s_i, e_i)$, $s_i < e_i \leq s_{i+1}$). Next, let $\mathscr{M}$ be the set of every possible motif, and let $T_p$, $S_t$, $R$, $I$, $A_r$ be one-to-one mappings (transformations) from $\mathscr{M}$ to $\mathscr{M}$, where $T_p$ is the transposition by pitch interval $p$, $S_t$ is the shift by time interval $t$ ($p, t \in \mathbb{R}$), $R$ is the retrograde, $I$ is the inversion, and $A_r (r > 0)$ is the $r$-fold argumentation (diminution, in the case of $0 < r < 1$). These transformations generate a transformation group $\mathscr{T}$ whose operation is the composition of two transformations and whose identity element is the transformation that does noting. Each transformation in $\mathscr{T}$ is a *strict imitation* that preserves the internal structures of the motifs.

Here, a binary relation between a motif $m$ ($\in \mathscr{M}$) and $\tau(m)$ ($\tau \in \mathscr{T}$) can be defined. Due to the group structure of $\mathscr{T}$, this relation is an equivalence relation (i.e., it satisfies reflexivity, symmetry and transitivity[6]). Then, it derives equivalence

classes in $\mathscr{M}$. Because the motifs that belong to a same equivalence class share the same internal structure, they can be regarded as identical (or the same type)[1].

## 3 Formulation as a Set Partitioning Problem

A set partitioning problem, which is well known in the context of operations research, is an optimization problem defined as follows. Let $N$ be a set that consists of $n$ elements $\{N_1, N_2, \cdots, N_n\}$, and let $M$ be a family of sets $\{M_1, M_2, \cdots, M_m\}$, where each $M_j$ is a subset of $N$. If $\bigcup_{j \in X} M_j = N$ is satisfied, $X$, a subset of indexes of $M$, is called a cover, and the cover $X$ is called a partition if $M_{j_1} \bigcap M_{j_2} = \emptyset$ is satisfied for different $j_1, j_2 \in X$. If a constant $c_j$ called a cost is defined for each $M_j$, the problem of finding a partition $X$ that minimizes the sum of the costs $\sum_{j \in X} c_j$ is called a set partitioning problem.

### *3.1 Condition of motif division*

If $N_i$ corresponds to each note of a musical piece to be analyzed and $M_j$ corresponds to a motif, the problem of finding the most economically efficient motif division can be interpreted as a set partitioning problem. The index $i$ starts from the first note of a voice to the last note of the voice and from the first voice to the last voice. $M_j (1 \leq j \leq m)$ corresponds to $[N_1]$, $[N_1, N_2]$, $[N_1, N_2, N_3]$, $\cdots$, $[N_2]$, $[N_2, N_3]$, $\cdots$ in this order. The number of notes in a motif is less than a certain limit number (Fig. 1).

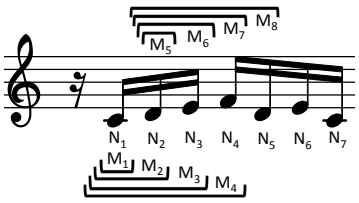



**Fig. 1** Possible motifs of the first voice of J.S. Bach's *Invention No. 1*. (In the case where the maximum number of notes in a motif is 4.

This information can be represented by the following matrix $A$:

[1] Although the criterion for identical motifs defined here only deals with strict imitations, we can define the criterion in different ways to allow more flexible imitations, such as by (1) defining an equivalence relation from the equality of a shape type [7, 8, 9, 10] and (2) defining a similarity measure and performing a clustering of motifs using methods such as k-medoid method [11] (the resulting clusters derive an equivalence relation). In any case, making equivalence classes from a certain equivalence relation is a versatile way to define the identicalness of the motifs.

$$A = \begin{bmatrix} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&\cdots \\ 0&1&1&1&1&1&1&1&0&0&0&0&0&0&\cdots \\ 0&0&1&1&0&1&1&1&1&1&1&1&1&0&\cdots \\ 0&0&0&1&0&0&1&1&0&1&1&1&1&1&\cdots \\ 0&0&0&0&0&0&0&1&0&0&1&1&0&\cdots \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&\cdots \\ \vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\vdots&\ddots \end{bmatrix} \tag{1}$$

where each row corresponds to each note $N_i$ and each column corresponds to which notes are covered by each motif $M_j$. This matrix is the case where the maximum number of notes in a motif is 4. Representing the element of $A$ as $a_{ij}$, the condition that the whole piece is exactly divided by a set of selected motifs can be described by the following constraints, which mean that each note $N_i$ is covered by one of $M_j$ once and only once:

$$\forall i \in \{1,2,\cdots,n\},\ \sum_{j=1}^{m} a_{ij}x_j = 1, \tag{2}$$

where $x_j$ is a 0-1 variable that represents whether or not $M_j$ is used in the motif division. These conditions are equivalent to the condition of partitioning.

### *3.2 Objective Function*

The purpose of motif division is to find the most efficient solution from the many solutions that satisfy the condition of partitioning. Then, we must define efficiency of motif division. We can consider that the average length (the number of notes) of motifs used in the motif division is one of the simplest barometers that represent the efficiency of motif division. Also, the number of motifs and that of the types of motif used in motif division will be efficient if they are small. In fact, the average length of motifs is inversely proportional to the number of motifs. Therefore, if the number of types of motif (denoted by $P$) is fixed, the number of motifs will be what we should minimize.

The number of motifs can be simply represented by $\sum_{j=1}^{m} x_j$ This is the cost function $\sum_{j=1}^{m} c_j x_j$ whose $c_j$ is 1 for each $j$. We adopt this cost function. However, in the next subsection, we introduce additional variables and constraints to fix the number $P$.

### *3.3 Controlling the Number of Equivalence Classes*

Let $C$ be the set of equivalence classes of motif, which is derived from $M$, which is the set of all possible motif classes that can be found in a piece (only the motif classes whose number of notes is less than a certain number is included in $M$).

This means that $M$ is derived from $\mathscr{M}$ by a restriction. Let $y_k$ be a 0-1 variable that represents whether or not one of the members of $C_k$ appears in $X$ (the set of selected motifs), where each element of $C$ is denoted as $C_k(1 \leq k \leq l)$. This means that statement "$y_k = 1 \Leftrightarrow \sum_{j \in C_k} x_j > 0$" must be satisfied. This statement can be represented by the following constraints that use $\sum_{j \in C_k} x_j$, the number of selected motifs that belong to $C_k$:

$$\forall k \in \{1, 2, \cdots, l\},\ y_k \leq \sum_{j \in C_k} x_j \leq Q y_k, \tag{3}$$

where $Q$ is a constant that is sufficiently large. Then, the statement that the number of equivalence classes is $P$ can be represented by the following constraint:

$$\sum_{k=1}^{l} y_k = P. \tag{4}$$

If $P$ is small to a certain degree, the motif division will tend to be simple. However, if $P$ is too small, covering whole piece with few motif classes will be difficult and one note motif will be used too many times. This will lead to a loss of the efficiency of motif division. Therefore, we should find good balance between the smallness of the objective function and the smallness of $P$. Because knowing which number is adequate for $P$ in advance is difficult, we will solve the optimization problems for respective $P$ in a certain range. Then, we will determine an adequate number for $P$, observing the solutions for respective $P$.

## 4 Result

We analyzed J.S. Bach's *Invention No. 1* by solving the optimization problem described in the previous section. The maximum length of motif was set as 7. An IP solver *Numerical Optimizer 16.1.0.* and a branch and bound method was used for searching the solution. From the observation of solutions for various values for $P$, $P$ was set as 13. It took less than one minute to obtain a solution for $P = 13$. Fig. 3 shows the result of motif division. The slurs represent the motifs and the one-note motifs don't have a slur. Fig. 3 shows the representatives of 13 motif classes that are used in the motif division.

This result tells us many things. For example, 4th, 10th, and 11th motif classes in Fig. 3 are slightly different but can be regarded as the same motif, which corresponds to the subject of this piece. Searching for the domains where the subject doesn't appear, we find that there are three domains whose durations are one and half bars (These are indicated by the big rectangles). The ends of these domains coincide with the places where the cadences exist. Therefore, we could detect three sections of this piece properly.

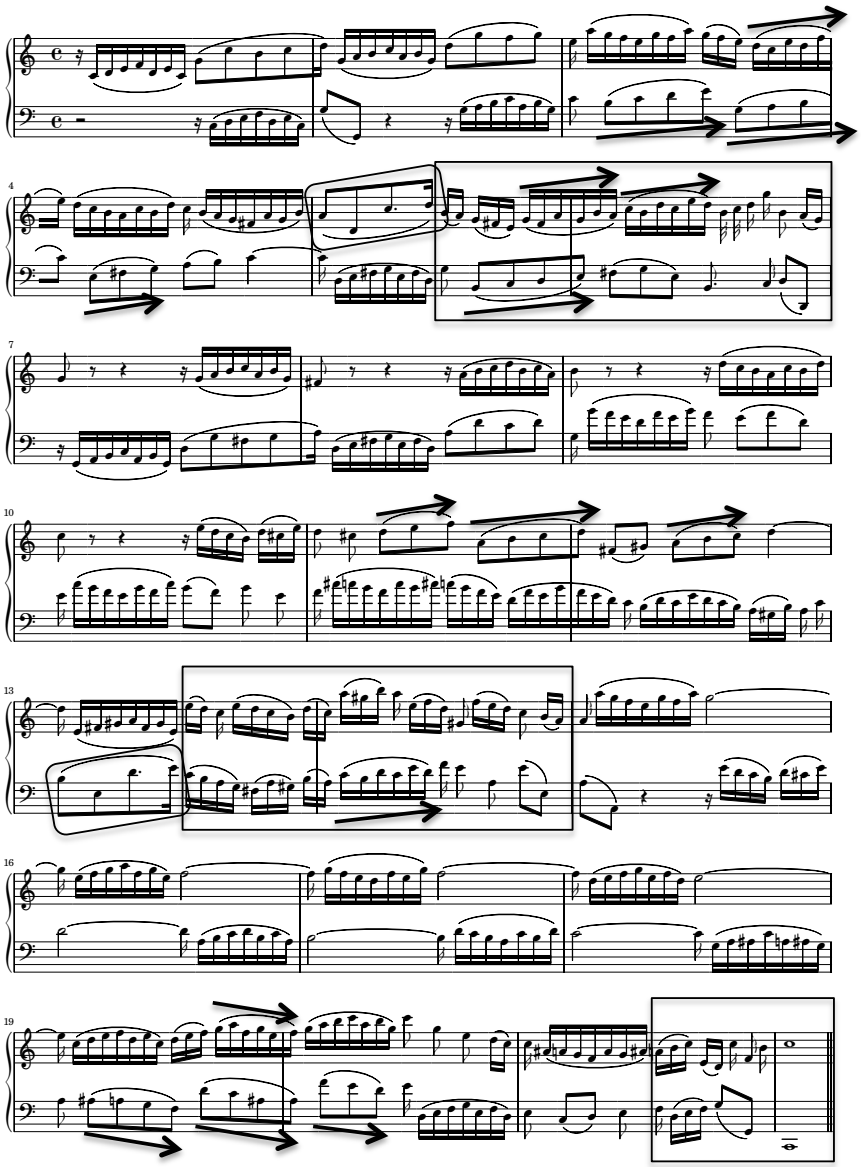


**Fig. 2** The representatives of motif classes that appear in the motif division of J.S. Bach's *Invention No. 1.* in the case that $P = 13$. Some flats are replaced by sharps for the purpose of programming.

The last motif class in Fig. 3 is a leap of octave. This motif class appears in all of the cadence domains and is related to the ends of sections. It also co-occurs with 2nd motif class, which is a two-note motif, in the cadence domains.

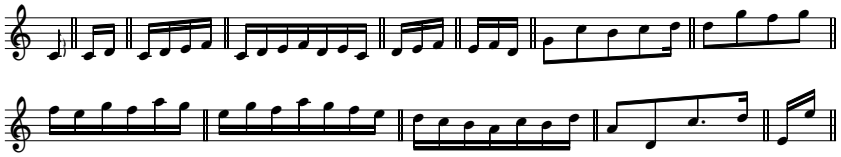


**Fig. 3** The motif classes that appear in the motif division of J.S. Bach's *Invention No. 1* (the number of motif classes was set at 13.).

The 12th motif class is a very characteristic one that includes a doted note and a large leap. This motif class only appears before the cadence domains (the two motifs surrounded by the rounded rectangle). We can consider that this remarkable motif class plays an important role that tells listeners the end of the exposition of subject and the beginning of the cadence domain.

The 9th zigzag motif class and the motif classes that are one-way slow movements shown by the arrows in Fig. 3 only appear as the ascending form in the first 2 sections. In contrast, these motif classes appear only as the descending form in the final section. We interpret this contrast means that the ascending form creates a sense of continuation of the piece and the descending form creates a sense of conclusion. Thus, long-term musical expectations seems to be formed by the selections of transformation.

In such ways, motif division is useful to make us understand the roles of motifs and how global musical structures are formed.

## 5 Conclusion

In this paper, we formulated the problem of motif division, which decompose polyphonic music into a small number of motif classes, as a set partitioning problem, and we obtained the solution using an IP solver. It was shown that the motif division provides useful information to understand the roles of motifs and how grobal musical structures are constructed from the motifs.

Future tasks include construction of a program that automatically analyzes global structures utilizing the obtained motifs and automatic composition of new pieces that use the same motifs as the original piece using the result of the analysis program. To create a criterion for determining adequate value of $P$ automatically is also a remaining problem.

**Acknowledgements** This work was supported by JSPS Postdoctoral Fellowships for Research Abroad.

## References

1. http://tetris.com
2. Cambouropoulos, E.: The Local Boundary Detection Model (LBDM) and its Application in the Study of Expressive Timing, Proc. ICMC, pp.290—293 (2001).
3. Ferrand, M., Nelson, P. and Wiggins, G.: Memory and Melodic Density: A Model for Melody Segmentation, Proc. the XIV Colloquium on Musical Informatics, pp.95—98 (2003).
4. Nemhauser, G.L. and Wolsey, L.A.: Integer and Combinatorial Optimization, Wiley (1988).
5. Balas, E. and Padberg, M.W.: Set Partitioning: A survey, SIAM Review 18.4, pp.710-760 (1976).
6. Armstrong, M.A.: Groups and Symmetry, Springer (2012).
7. Buteau, C and Mazzola G.: From Contour Similarity to Motivic Topologies, Musicae Scientiae Fall 2000 vol.4, no.2, pp.125-149 (2000).
8. Mazzola, G., et al.: The Topos of Music: Geometric Logic of Concepts, Theory, and Performance, Birkhäuser (2002).
9. Buteau, C.: Topological Motive Spaces, and Mappings of Scores Motivic Evolution Trees. In: Fripertinger, H., Reich, L. (eds.) Grazer Mathematische Berichte, Proceedings of the Colloqium on Mathematical Music Theory, pp.27—54 (2005)
10. Buteau, C.: Melodic Clustering Within Topological Spaces of Schumann ' s Träumerei, Proceedings of ICMC, pp.104-110 (2006).
11. Bishop, C.: Pattern Recognition and Machine Learning, Springer, pp.423-430 (2006).

# Describing Global Musical Structures by Integer Programming on Musical Patterns

Tsubasa Tanaka[1] and Koichi Fujii[2]

[1]Institut de Recherche et Coordination Acoustique/Musique, Paris, France
tsubasa.tanaka@ircam.fr
[2]NTT DATA Mathematical Systems Inc., Tokyo, Japan
fujii@msi.co.jp

**Abstract.** Music can be regarded as sequences of localized patterns, such as chords, rhythmic patterns, and melodic patterns. In the study of music generation, how to generate musically adequate sequences is an important issue. In particular, generating sequences by controlling the relationships between local patterns and global structures is a difficult and open problem. Whereas the grammatical approaches that represent global structures are suitable to analyze how the pieces are constructed, they are not necessarily designed to generate new pieces with controlling their characteristics of global structures such as the redundancy of the sequence and the statistical distribution of specific patterns. To achieve this, we must overcome the difficulty of solving computationally complex problems. In this paper, we take an integer-programming-based approach and show that some important characteristics of global structures can be described only by linear equalities and inequalities, which are suitable for the integer programming.

**Keywords:** musical patterns, global structure, hierarchy, redundancy, integer programming

## 1 Introduction

Music can be regarded as sequences of localized musical patterns such as chords, rhythmic patterns, and melodic patterns. In the study of music generation, it is important to know the characteristics of these sequences. For example, a Markov model is used to learn transition probabilities of localized musical elements in existing pieces or real-time performances [1], and pieces that imitate the original styles are expected to be generated from the model.

However, learning local characteristics is not sufficient to understand or generate music. Global musical structures or musical forms are necessary to be considered. Contrary to Markov models, grammatical approaches such as generative theory of tonal music [2] (GTTM, hereafter), are used to analyze the global structures of musical pieces. In GTTM, a musical piece is abstracted step by step by discarding less important elements and the whole piece is understood as a hierarchical tree structure. For example, Hamanaka et al. have implemented

GTTM on a computer and analyzed musical pieces [3]. However, this model does not have a strategy for composing new pieces. We should note that the grammatical models are not designed to generate new pieces with specifying their characteristics of global structures such as the redundancy of the sequence and the statistical distribution of specific patterns that actually appear in the generated pieces.

Therefore, the objective of this paper is to propose a model that can give specifications of characteristics of global musical structures in the context of music generation. We expect the proposed model to be applied to help users generate new pieces of music and/or obtain desired musical structures.

Compared to the problem of analysis, the problem of generation has a difficulty of a combinatorial explosion of possibilities. When we compose a new piece or sequence, we have to choose a sequence from all the possible combinations of patterns, whose number increases exponentially depending on the number of basic patterns. To deal with such combinatorial problems, we propose an integer-programming-based approach. In order to apply the integer programming, the structural specifications have to be described by linear equalities and inequalities. This is the main challenge for our study.

Integer programming is a framework to solve linear programming problems whose variables are restricted to integer variables or 0–1 variables [4]. Although it shares something in common with constraint programming, which has been often used in the realm of music, one of the advantages of integer programming is that it has an efficient algorithm to find the optimum solution by updating the estimations of the lower and upper bounds of the optimum solution based on the linear programming relaxation technique. Integer programming has been applied to various problems. In our previous study [5], musical motif analysis was formulated as a set partitioning problem, which is a well-known integer programming problem. Thanks to the recent improvements of integer programming solvers such as Numerical Optimizer [6], more and more practical problems have been solved within a reasonable time. We expect that integer programming may also play an important role in the generation of music.

Other possible generation methods than integer programming and constraint programming include the use of metaheuristics. In the study of solving counterpoint automatically [7], counter melodies were generated based on the local search whose objective function is defined as how well the generated counter melody satisfies the rules of counterpoint. In such a search algorithm, there is no guarantee to find the optimum solution. Especially, in the case where there are conflicts between many rules, some rules might be violated. This is not preferable for us because we think that the structural specification should be strictly respected. Therefore, we take an integer-programming approach, which we think is suitable to find strict solutions for the discrete optimization problems.

There are several limitations about what we can describe in this paper. Three main limitations are as follows: (1) we do not describe the objective function and only focus on the constraints to be satisfied strictly (however, we are planning to introduce the objective function in the future work). (2) We do not propose

a comprehensive formulation that covers various situations. We would rather explain our point of view based on a concrete examples of typical musical structure. (3) We only focus on the aspect of formulation and do not proceed to the steps of generation and evaluation.

The rest of this paper is organized as follows: In section 2, we show the relationship between localized musical patterns and hierarchical global structure referring to the chord progression of a piece that has typical phrase structures. In section 3, we describe how to implement this relationship by linear equalities and inequalities. In section 4 and section 5, we describe the similar relationships for the structures of rhythmic pattern sequence and intervallic pattern sequence by linear constraints. Then, these are combined to formulate the constraints for generating a melody. Section 6 gives some concluding remarks.

## 2 Hierarchy and Redundancy

From the micro-level point of view, composing a piece of music can be regarded as determining how the sequences of localized musical patterns are arranged. If such sequences are seen from the macro-level point of view, they can be regarded as the musical forms.

Then, a question arises: what are the global characteristics that a musical sequence should have to construct a global musical form and to generate well-organized music? In this section, we study the piece Op. 101 No. 74 by Ferdinand Beyer to consider this problem. The chord sequence of this piece is clearly related to the phrase structures and the musical form, and it will provide a good clue for this problem.

The chord sequence of this piece is TTSTTTDTSTSTTTDTDTDTDTDT, using T (tonic), S (subdominant), and D (dominant). Each chord corresponds to one measure. To see the phrase structures more clearly, let's combine each two bars, four bars, and eight bars. Then these four levels can be represented as the following sequences:

- $1^{st}$ level: $A_1A_1A_2A_1A_1A_1A_3A_1A_2A_1A_2A_1A_1A_1A_3A_1A_3A_1A_3A_1A_3A_1A_3A_1$ (3/24)
- $2^{nd}$ level: $B_1B_2B_1B_3B_2B_2B_1B_3B_3B_3B_3B_3$ (3/12)
- $3^{rd}$ level: $C_1C_2C_3C_2C_4C_4$ (4/6)
- $4^{th}$ level: $D_1D_2D_3$ (3/3)

Here, the same indices indicate the same patterns. For example, $A_1$ =T, $A_2$ =S and $A_3$ =D. There are relationships between the consecutive levels such as $B_1 = A_1A_1$, $C_1 = B_1B_2$, $D_1 = C_1C_2$, and so on (The notations like "$A_1A_1$" represent the concatenations of the patterns and do not indicate multiplications). The numerators of the fractions indicated between parentheses after each sequence indicate the numbers of the variety of patterns that exist in the sequences in each level, and the denominators indicate the lengths of each level. We call the inverse numbers of these fractions redundancies[1]. Fig. 1 visualizes

[1] The study [8] focuses on the redundancy of musical sequence, and models the musical style by referring to the Lempel-Ziv compression algorithm. We extend this perspective and pay attention to the redundancies of multiple levels simultaneously.

this hierarchical structure. Because the fourth level has three different elements,

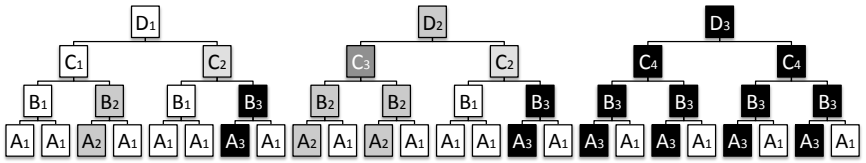



**Fig. 1.** Hierarchical structure of chord sequence of Beyer No. 74.

we see that this piece consists of three different parts. When observing the relationship between the third level and fourth level, we find that the latter parts of $D_1$ and $D_2$ are the same and $D_3$ consists of a repetition of $C_4$.

We can observe that there are many repetitions of the same patterns and that the lower the level, the higher the redundancy. In addition, T is more than twice as frequent as D, and D is twice as frequent as S. These biases of frequency are also notable characteristics of this sequence. Thus, the global structure of this piece can be regarded to be, at least, constrained by hierarchy, redundancies, and frequencies of patterns. Without repetitions in lower levels, the music will become unmemorable and lose the attention of human listeners. Moreover, repetitions in higher levels are related to known musical forms such as A-B-A ternary form or rondo form. Thus, frequency and redundancy are important fearures for describing musical structures.

Focusing on each level, state transition diagrams can be depicted per level (Fig. 2). From the global point of view, state transitions of multiple levels should

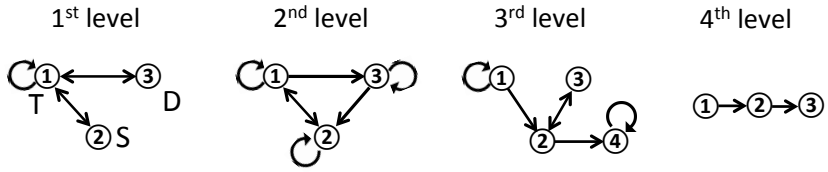



**Fig. 2.** Transition diagram of chord patterns in each level.

be considered simultaneously. In these diagrams, the lowest level (first level) represents the rules of chord progression proper to this piece. Although the transition from S to D is possible in ordinary rules of harmony, this transition is not used in this piece.

## 3 Implementation of Constraints on Chord Progression

It is difficult to find new sequences that have specific redundancies in respective levels, because the constraints for different levels may cause conflicts during the process of searching for the solutions. Also, the naive method of enumerating possible sequences one by one and checking whether or not they satisfy specific

redundancies in each level won't be realistic because the number of possible sequences increases exponentially with the length of the sequence based on the number of the states in the lowest level.

To deal with this problem, we take an integer-programming-based approach, which has good search algorithms to practically solve computationally complex problems. In order to use integer programming, the characteristics of the sequence should be expressed by linear constraints (equalities and inequalities). In this section, we demonstrate that some important characteristics of the sequence observed in the Beyer's piece can be expressed by linear constraints.[2]

Let $A$, $B$, $C$, $D$ be the set of patterns that can appear in respective levels. The number of elements in each level is denoted by $K_1$, $K_2$, $K_3$, and $K_4$. For explanation, we set these possible patterns as the patterns that actually appear in the Beyer's piece (i.e., $A = \{A_1, A_2, \cdots A_3\}$, $B = \{B_1, B_2, \cdots, B_3\}$, $\cdots$.). [3] Let $a_t$ be a variable that corresponds to the $t$-th element in the sequence of variables for the patterns of the first level, $b_t$ for those of the second level, $c_t$ for those of the third level, and $d_t$ for those of the fourth level. For example, in the original piece, $a_1 = A_1$, $a_2 = A_1$, $a_3 = A_2$, $a_4 = A_1$, $\cdots$, $a_{24} = A_1$, $b_1 = B_1$, $b_2 = B_2$, $b_3 = B_1$, $b_4 = B_2$, $\cdots$, $b_{12} = B_3$, and so on.

The followings are important characteristics of the original sequence explained in Section 2:

1. Hierarchy of phrase structures (relationships between neighboring levels): $b_t = a_{2t-1}a_{2t} (1 \leq t \leq 12)$, $c_t = b_{2t-1}b_{2t} (1 \leq t \leq 6)$, $d_t = c_{2t-1}c_{2t} (1 \leq t \leq 3)$.
2. Variety of patterns that appear in each level: the first level has three patterns, the second has three patterns, the third has four patterns, and the fourth has three patterns.
3. Frequency of each state in each level (in the first level, these are the frequency of each chord).
4. State transition rules (in the first level, these are the chord progression rules).

In this section, we show how to formalize these rules by variables and linear constraints on the variables.

### 3.1 Constraints of Hierarchical Phrase Structures

Let $x_{t,i}$, $y_{t,i}$, $z_{t,i}$, and $w_{t,i}$ be 0–1 variables that represent whether or not the $t$-th element of the sequence for each level is the pattern $i$ (For example, the pattern

[2] In this paper, in order to avoid the explanation from being complicated, we only treat the stereotype examples of musical pieces whose groupings are always combinations of two consecutive elements in every level. However, in practice, such structures should vary depending on the specifications of the pieces that the user wants to create. For example, we can think of the case where the number of combination in the groupings are different between the levels. We can also think of the case where the consecutive patterns can be overlapped as is mentioned in [9]. How to formulate such cases is an important future issue.

[3] In practice, it is not necessary to stick to existing pieces.

$i$ indicates $A_i$, in the case of the first level). $x_{t,i}$, $y_{t,i}$, $z_{t,i}$, and $w_{t,i}$ correspond to the first, second, third, and fourth levels respectively. For example, $x_{t,i} = 1$ if $a_t = A_i$, and otherwise $x_{t,i} = 0$. $y_{t,i} = 1$ if $b_t = B_i$ and otherwise $y_{t,i} = 0$. $z_{t,i}$ and $w_{t,i}$ are also defined similarly. $a_t$, $b_t$, $c_t$, and $d_t$ take one of the values in $A$, $B$, $C$, and $D$, respectively. The numbers of elements of $A$, $B$, $C$, and $D$ are denoted by $K_1$, $K_2$, $K_3$, and $K_4$, respectively. Because $a_t$, $b_t$, $c_t$, and $d_t$ take only one value at a location $t$, respectively (here, location $t$ means $t$-th element in the sequence of each level), they satisfy the following constraints:

$$\sum_{i=1}^{K_1} x_{t,i} = 1 \ (\forall t, 1 \le t \le 24), \quad \sum_{i=1}^{K_2} y_{t,i} = 1 \ (\forall t, 1 \le t \le 12),$$
$$\sum_{i=1}^{K_3} z_{t,i} = 1 \ (\forall t, 1 \le t \le 6), \quad \sum_{i=1}^{K_4} w_{t,i} = 1 \ (\forall t, 1 \le t \le 3). \tag{1}$$

Now the hierarchy can be expressed by constraints on $x_{t,i}$, $y_{t,i}$, $z_{t,i}$, and $w_{t,i}$. For example, if $B_i = A_{j_1}A_{j_2}$, the statement "$b_t = B_i$ (i.e. $y_{t,i} = 1$)" must be equivalent to "$a_{2t-1} = A_{j_1}$ and $a_{2t} = A_{j_2}$ (i.e., $x_{2t-1,j_1} = 1$ and $x_{2t,j_2} = 1$)", and this equivalence can be expressed by the three constraints: $y_{t,i} \le x_{2t-1,j_1}$, $y_{t,i} \le x_{2t,j_2}$, and $x_{2t-1,j_1} + x_{2t,j_2} - 1 \le y_{t,i}$. Therefore, the constraints that correspond to the whole hierarchy are described as follows:

for all $(i, j_1, j_2)$ that satisfy $B_i = A_{j_1}A_{j_2} (1 \le i \le K_1)$,

$$y_{t,i} \le x_{2t-1,j_1},\ y_{t,i} \le x_{2t,j_2},\ x_{2t-1,j_1} + x_{2t,j_2} - 1 \le y_{t,i} \ (1 \le t \le 12), \tag{2}$$

for all $(i, j_1, j_2)$ that satisfy $C_i = B_{j_1}B_{j_2} (1 \le i \le K_2)$,

$$z_{t,i} \le y_{2t-1,j_1},\ z_{t,i} \le t_{2t,j_2},\ y_{2t-1,j_1} + y_{2t,j_2} - 1 \le z_{t,i} \ (1 \le t \le 6), \tag{3}$$

for all $(i, j_1, j_2)$ that satisfy $D_i = C_{j_1}C_{j_2} (1 \le i \le K_3)$,

$$w_{t,i} \le z_{2t-1,j_1},\ w_{t,i} \le z_{2t,j_2},\ z_{2t-1,j_1} + z_{2t,j_2} - 1 \le w_{t,i} \ (1 \le t \le 3). \tag{4}$$

Finally, the statement that every element in the last level is different can be expressed by the following constraints:

$$w_{1,i} + w_{2,i} + w_{3,i} \le 1 \ (\forall i, 1 \le i \le K_4). \tag{5}$$

### 3.2 Constraints on Frequencies of Each Patterns

The constraints to control the frequency (or the range of frequency) of each state can be described by the following inequalities using 0–1 variables $\alpha_i$, $\beta_i$, $\gamma_i$, and $\delta_i$ for each level:

$$L_{1,i}\alpha_i \le \sum_{t=1}^{24} x_{t,i} \le H_{1,i}\alpha_i \ (\forall i, 1 \le i \le K_1), \tag{6}$$

$$L_{2,i}\beta_i \leq \sum_{t=1}^{12} y_{t,i} \leq H_{2,i}\beta_i \ (\forall i, 1 \leq i \leq K_2), \tag{7}$$

$$L_{3,i}\gamma_i \leq \sum_{t=1}^{6} z_{t,i} \leq H_{3,i}\gamma_i \ (\forall i, 1 \leq i \leq K_3), \tag{8}$$

$$L_{4,i}\delta_i \leq \sum_{t=1}^{3} w_{t,i} \leq H_{4,i}\delta_i \ (\forall i, 1 \leq i \leq K_4), \tag{9}$$

where $L_{1,i}$, $L_{2,i}$, $L_{3,i}$, $L_{4,i}$, $H_{1,i}$, $H_{2,i}$, $H_{3,i}$, and $H_{4,i}$ are the constants[4] for lower and upper bounds in the case that the $i$-th element of each level appears. The statement "$\alpha_i = 0$" is equivalent to "$x_{t,i} = 0$ for all $t(1 \leq t \leq 24)$". This means that $\alpha_i$ represents whether or not the state $a_i$ appears in the sequence. $\beta_i$, $\gamma_i$, and $\delta_i$ also have such meanings in their own levels.

Also, relative differences of frequencies that T is more than twice or twice as frequent as D and D is twice as frequent as S can be expressed by the following constraints:

$$\sum_{t=1}^{24} x_{t,1} \geq 2\sum_{t=1}^{24} x_{t,3}, \quad \sum_{t=1}^{24} x_{t,3} = 2\sum_{t=1}^{24} x_{t,2} \tag{10}$$

### 3.3 Constraints on Varieties of Patterns

Using the variables $\alpha_i$, $\beta_i$, $\gamma_i$, and $\delta_i$, the number of variety of patterns in each level can be described. For example, $\sum_i \alpha_i$ indicates the number of variety of patterns in the first level. Similarly, the statement that the numbers of variety of patterns in respective levels are 3, 3, 4, and 3 can be represented as:

$$\sum_{i=1}^{K_1} \alpha_i = 3, \quad \sum_{i=1}^{K_2} \beta_i = 3, \quad \sum_{i=1}^{K_3} \gamma_i = 4, \quad \sum_{i=1}^{K_4} \delta_i = 3. \tag{11}$$

### 3.4 Constraints on State Transitions

The possibility of state transitions in the first level can be controlled by posing the following inequality for all of the combinations of $(t, i, j)$ whose transition from $i$ to $j$ at the location $t$ is prohibited:

$$x_{t,i} + x_{t+1,j} \leq 1. \tag{12}$$

The same is true in other levels. One way to determine the allowed transitions is to prohibit the transitions that do not occur in the original piece.

Thus, linear constraints can describe both global structures and local state transitions.

[4] For example, they can be set depending on user's preference or statistics of the original piece. If $L_{j,i} \leq 1$ and $H_{j,i}$ is larger than or equal to the length of the sequence, these equations give no limitation to the number of each pattern that appears in the sequence.

## 4 Constraints on Rhythmic Patterns

In this section, the hierarchical structure of rhythmic patterns is illustrated in a similar way to the hierarchical structure of chords. Beethoven's famous melody *Ode to Joy* is used as an model example of monophony (Fig. 3).

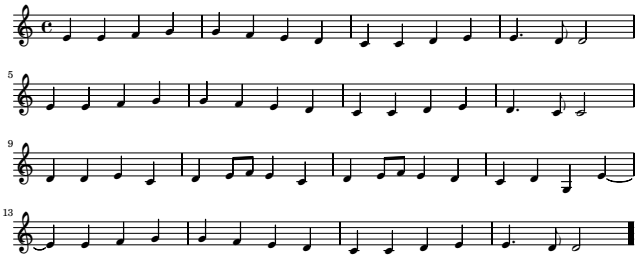


**Fig. 3.** Beethoven's melody *Ode to Joy*.

Adopting a beat as a unit length, the different elements of the first level are $A_1 = [f]$, $A_2 = [f\sim]$, $A_3 = [\sim e, e]$, $A_4 = [\sim f]$, and $A_5 = [e, e]$, where $e$, $f$, and $s$ represent 8th note, 4th note, 2nd note, respectively, and "." and "$\sim$" represent a "dot" and a "tie," respectively. Each $A_i$ represents one beat and does not necessarily correspond to the actual notations (the "ties" and "dots" actually correspond to the ties or prolongations of a note beyond the beat). The second level consists of $B_1 = [f, f]$, $B_2 = [f., e]$, $B_3 = [s]$, $B_4 = [f, e, e]$, $B_5 = [f, f\sim]$, and $B_6 = [\sim f, f]$. The entire hierarchy is as follows:

- $a_t$ ($1^{st}$ level): $||A_1A_1A_1A_1|A_1A_1A_1A_1|A_1A_1A_1A_1|A_2A_3A_2A_4|| \cdots$ (5/64)
- $b_t$ ($2^{nd}$ level): $||B_1B_1|B_1B_1|B_1B_1|B_2B_3|| \cdots$ (6/32)
- $c_t$ ($3^{rd}$ level): $||C_1|C_1|C_1|C_2||C_1|C_1|C_1|C_2||C_1|C_3|C_3|C_4||C_5|C_1|c_1|c_2||$ (5/16)
- $d_t$ ($4^{th}$ level): $||D_1D_2||D_1D_2||D_3D_4||D_5D_1||$ (5/8)
- $e_t$ ($5^{th}$ level): $||E_1||E_1||E_2||E_3||$ (3/4)
- $f_t$ ($6^{th}$ level): $||F_1\ F_2||$ (2/2)

where, "|" is a bar line and "||" is a four-bar partition. This hierarchy can be described by linear constraints in a similar way to the previous section. However, this case has more levels than the previous case. In general, the number of possible patterns increases drastically when looking at a higher level. If the number of levels is too large, the number of variables for the level may become too large.

Therefore, we introduce an alternative way to describe the constraints for redundancies of higher levels using the variables for the first level $x_{t,i} (1 \leq i \leq K_1)$ (without using $y_{t,i}$, $z_{t,i}$, and $w_{t,i}$).[5]

[5] However, at the current moment, we do not know an alternative way to implement the constraints for state transitions and frequencies of each pattern in the high levels.

### 4.1 Alternative Way to Control Redundancies of High Levels

Let us consider a 0–1 variable $s_{n,t}$ whose value is 1 if and only if an element in a level $n$ first appear at the location $t$. Then, $\sum_t s_{n,t}$ represents how many different elements appear in level $n$. This offers an alternative way to specify the redundancies. Our purpose here is to construct the variables $s_{n,t}$ that have such meaning. In order to do so, we introduce other subsidiary variables $q_{n,t_1,t_2}$ and $r_{n,i,u1,u2}$.

$q_{n,t_1,t_2}$ is a 0–1 variable that represents whether the sections $t_1$ and $t_2$ ($t_1 < t_2$) in level $n$ are the same or not (we call each content of the time span of an element such as $b_t$, $c_t$, $\cdots$ a section). $q_{n,t_1,t_2} = 1$ means that all of corresponding elements $a_{u_1}$ and $a_{u_2}$ in the sections $t_1$ and $t_2$ in level $n$ are the same. Therefore, it is necessary to introduce the constraints that make the statement "$q_{n,t_1,t_2} = 1 \Longleftrightarrow x_{u_1,i} = x_{u_2,i}$ $(\forall(i, u_1, u_2)$ s.t. $i \in [1, K_n]$, $(u_1, u_2) \in D(n, t_1, t_2))$" true, where $D(n, t_1, t_2)$ represents the range of the combinations $(u_1, u_2)$ where $a_{u_1}$ and $a_{u_2}$ are all of the pairs of corresponding elements in the sections $t_1$ and $t_2$ in level $n$. This statement can be replaced by the statement "$r_{n,i,u_1,u_2} = 1$ $(\forall(i, u_1, u_2)$ s.t. $i \in [1, K_n]$, $(u_1, u_2) \in D(n, t_1, t_2)) \Longleftrightarrow q_{n,t_1,t_2} = 1$," where the 0–1 variable $r_{n,i,u_1,u_2}$ means whether $x_{u_1,i} = x_{u_2,i}$ or not.

Here, the statement "$r_{n,i,u_1,u_2} = 1 \Longleftrightarrow x_{u_1,i} = x_{u_2,i}$," can be expressed by the following constraints:

$$r_{n,i,u_1,u_2} \leq 1 + x_{u_1,i} - x_{u_2,i}, \tag{13}$$

$$r_{n,i,u_1,u_2} \leq 1 - x_{u_1,i} + x_{u_2,i}, \tag{14}$$

$$1 - x_{u_1,i} - x_{u_2,i} \leq r_{n,i,u_1,u_2}, \tag{15}$$

$$-1 + x_{u_1,i} + x_{u_2,i} \leq r_{n,i,u_1,u_2}. \tag{16}$$

Also, the statement "$r_{n,i,u_1,u_2} = 1$ $(\forall(i, u_1, u_2)$ s.t. $i \in [1, K_n]$, $(u_1, u_2) \in D(n, t_1, t_2)) \Longleftrightarrow q_{n,t_1,t_2} = 1$," can be expressed by the following constraints:

$$1 - \sum_{i \in [1,K_n]} \sum_{(u_1,u_2) \in D(n,t_1,t_2)} (1 - r_{n,i,u1,u2}) \leq q_{n,t_1,t_2}, \tag{17}$$

$$q_{n,t_1,t_2} \leq r_{n,i,u_1,u_2} \quad (\forall(i, u_1, u_2)\ s.t.\ i \in K_n,\ (u1, u2) \in D(n, t_1, t_2)). \tag{18}$$

Let $v_{n,t}$ be $\sum_{t_1<t} q_{n,t_1,t}$, then $v_{n,t}$ represents how many the same sections as $t$ there are before the section $t$ in level $n$. $v_{n,t} = 0$ means that the content of section $t$ first appears. Therefore, the statement "$v_{n,t} = 0 \Longleftrightarrow s_{n,t} = 1$" must be true to let $s_{n,t}$ have the proper meaning. This statement can be expressed by a constraint:

$$1 - s_{n,t} \leq v_{n,t} \leq M \cdot (1 - s_{n,t}), \tag{19}$$

where $M$ is a sufficiently large constant number. Under the constraints above, we can control the redundancy of each level $n$ by a constraint:

$$\sum_t s_{n,t} = Constant. \tag{20}$$

## 5 Constraints on Melodic Patterns

In this section, hierarchical structure of the melodic patterns, which has information of pitches and rhythms, is illustrated using the same melody *Ode to Joy*. Although pitch information and rhythmic information in melodic patterns are not completely independent of each other, the hierarchical structure of pitch patterns and rhythmic patterns are different in general. Therefore, we treat these two hierarchical structures separately. After that, we consider compatibility between them and combine them. Considering that the same pitch patterns can occur in different transpositions on the scale, we treat pitch information as the series of intervals. In section 5.1, we introduce constraints on structure of intervallic patterns. In section 5.2, constraints for controlling pitch range is introduced. Then, in section 5.3, we describe how to combine the rhythmic structure and the intervallic structure to generate melodies.

### 5.1 Constraints on Intervallic Patterns

There are nine different one-beat intervallic patterns that appear in the piece *Ode to Joy*. The intervals are denoted by the interval numbers on the scale - 1, based on the diatonic scale[6]. These are [0], [1], [−1], [∼], [−1, 0], [−2], [1, −1], [−4], [5], which are denoted by $A_1 \sim A_9$, respectively. These are series of intervals that start from the interval between the first pitch and the second pitch of the beat to the interval between the last pitch to the first pitch of the next beat. [∼] means the interval 0 by a tie or a prolongation of a pitch to the next pattern. Though there is no beat after the last beat, the last beat obviously corresponds to the last beats of the bar 4 and bar 8. Therefore, we regard the last beat as the same intervallic pattern as the last beats of the bar 4 and bar 8, namely [1]. The sequences of each level is as follows:

- $a_t$ ($1^{st}$ level): $||A_1A_2A_2A_1|A_3A_3A_3A_3|A_1A_2A_2A_1|A_4A_5A_4A_1|| \cdots$ (9/64)
- $b_t$ ($2^{nd}$ level): $||B_1B_2|B_3B_3|B_1B_2|B_4B_5|| \cdots$ (10/32)
- $c_t$ ($3^{rd}$ level): $||C_1|C_2|C_1|C_3||C_1|C_2|C_4|C_3||C_5|C_6|C_7|C_8||C_1|C_2|C_1|C_3||$ (8/16)
- $d_t$ ($4^{th}$ level): $||D_1D_2||D_1D_3||D_4D_5||D_1D_2||$ (5/8)
- $e_t$ ($5^{th}$ level): $||E_1||E_2||E_3||E_1||$ (3/4)
- $f_t$ ($6^{th}$ level): $||F_1\ F_2||$ (2/2)

Comparing, for example, the sequence of the fifth level of the rhythmic patterns $||E_1||E_1||E_2||E_3||$) and the sequence of the intervallic patterns ($||E_1||E_2||E_3||E_1||$), we see that the first two elements are the same and the first and last elements are not the same in the former sequence. On the other hand, the first two elements are not the same and the first and the last elements are the same in the latter sequence. This difference adequately represents that although the first, second, and fourth 4-bars phrases are almost the same, the first and second have

[6] Here, we can also represent the pitches and the intervals based on the chromatic scale. However, we use the scale degrees and the interval numbers based on the diatonic scale because that is more efficient.

the same rhythms but slightly different intervals and the first and last 4-bars phrases have the same intervals but slightly different rhythms. This is the benefit of treating the rhythmic structure and the intarvallic structure separately.

### 5.2 Constraints for Pitch Range

The intervallic structures described in the previous section do not have the limitation of the pitch range. This problem occurs because the intervallic patterns are based on relative intervals instead of absolute pitches. Therefore, in this subsection, we introduce extra variables and constraints for bounding the pitch range.

Let $p_t$ be the first pitch of the pattern at the location $t$ of the sequence. The pitch is counted on the scale, where the starting pitch $p_1$ is set as 0. In the case of *Ode to Joy*, C $= -2$, D $= -1$, E $= 0$, etc. The scale of the piece is {-5, -4, -3, -2, -1, 0, 1, 2}. Let $Intvl(A_i)$ be the total interval of the intervallic pattern $A_i$ (e.g., $Intvl(A_7) = 0$, because the total interval of $A_7( = [1, -1])$ is 0 $(= 1 - 1)$). Then, $p_t$ is equal to the accumulation of the total intervals from the first pattern to the $(t-1)$-th pattern as in the following equation:

$$p_t = \sum_{t_1<t} Intvl(a_{t_1}) = \sum_{t_1<t} \sum_i Intvl(A_i) \cdot x_{t_1,i} \tag{21}$$

Let $Int(i, j)$ be the total interval from the beginning of $A_i$ to the end of $j$th interval. If $a_t = A_i$, the pitch after the $j$th interval of $A_i$ in $a_t$ is $p_t + Int(i, j)$. If the next constraint:

$$LB_t \leq p_t + Int(i, j)x_{t,i} \leq UB_t \tag{22}$$

is satisfied for all $(t, i, j)$, the upper bound $UB_t$ and lower bound $LB_t$ for pitches in $t$-th element of the sequence can be set.

### 5.3 Compatibility of Rhythmic Pattern and Intervallic Pattern

Although the sequences of rhythmic patterns and intervallic patterns have been independently introduced in the previous subsections, it is necessary to combine these two sequences to complete a melody, which contains both rhythms and intervals. To combine both of the sequences, compatibility between rhythmic patterns and intervallic patterns must be assured. For example, [s] and [-1,0] are not compatible because the number of elements differs ([s] indicates that there is only one note in this rhythmic pattern, and [-1,0] indicates that there are two notes in the intervallic pattern).

Therefore, we introduce constraints to assure that both sequences can coexist and propose a formulation to generate both sequences simultaneously. Let's discriminate variables and constants for intervallic patterns from those of rhythmic patterns by adding a dash on the shoulder of the variable names. The compatibility between both sequences depends on whether $a_t$ and $a'_t$ are compatible in

every location $t$ or not. Therefore, the compatibility can be expressed by the following inequality for all combination $A_{i_1}$ and $A'_{i_2}$ that are not compatible:

$$x_{t,i_1} + x'_{t,i_2} \leq 1. \tag{23}$$

If we find a solution for the problem that is a combination of the problems for rhythmic patterns and intervallic patterns with these inequalities, we will be able to obtain a complete melody.

## 6 Conclusion

In this paper, we proposed a formulation to generate sequences of musical patterns controlling some global structures of the sequences especially focusing on hierarchy and degree of redundancy in each level. We showed that such structures can be expressed only by linear constraints, which are necessary to apply integer programming. Future tasks include describing global structures more comprehensively, implementation of the constraints and actually generating new pieces, defining constraints on the relationships between melody and chords, and defining the constraints for polyphonic relationships.

**Acknowledgments.** This work was supported by JSPS Postdoctoral Fellowships for Research Abroad.

## References

1. Pachet, F.: "The Continuator: Musical Interaction With Style," Journal of New Music Research, Vol.32, No.3, pp.333-341 (2003).
2. Lerdahl, F., and Jackendoff, R.: A Generative Theory of Tonal Music, MIT Press (1983).
3. Hamanaka, M., et al.: "Implementing "A Generative Theory of Tonal Music,"" Journal of New Music Research, Vol.35, No.4, pp.249-277 (2006).
4. Nemhauser, G.L. and Wolsey, L.A.: Integer and Combinatorial Optimization, Wiley (1988).
5. Tanaka, T. and Fujii K.: "Melodic Pattern Segmentation of Polyphonic Music as a Set Partitioning Problem," Proceedings of International Congress on Music and Mathematics (to be published).
6. http://msi.co.jp/nuopt/
7. Herremans, D. and Srensen, K.: "A variable neighbourhood search algorithm to generate first species counterpoint musical scores," Working Paper, University of Antwerp Faculty of Applied Economics Operations Research Group ANT/OR, (2011).
8. Lartillot, O., et al.: "Automatic Modeling of Musical Style," 8èmes Journées d'Informatique Musicale, pp.113-119 (2001).
9. Mazzola, G., et al.: The Topos of Music: Geometric Logic of Concepts, Theory, and Performance, Birkhäuser (2002).

# INTEGER PROGRAMMING FORMULATION OF THE PROBLEM OF GENERATING MILTON BABBITT'S ALL-PARTITION ARRAYS

**Tsubasa Tanaka**
STMS Lab : IRCAM, CNRS, UPMC
Paris, France
tsubasa.tanaka@ircam.fr

**Brian Bemman**
Aalborg University
Aalborg, Denmark
bb@create.aau.dk

**David Meredith**
Aalborg University
Aalborg, Denmark
dave@create.aau.dk

## ABSTRACT

Milton Babbitt (1916–2011) was a composer of twelve-tone serial music noted for creating the *all-partition array*. The problem of generating an all-partition array involves finding a rectangular array of pitch-class integers that can be partitioned into regions, each of which represents a distinct integer partition of 12. Integer programming (IP) has proven to be effective for solving such combinatorial problems, however, it has never before been applied to the problem addressed in this paper. We introduce a new way of viewing this problem as one in which restricted overlaps between integer partition regions are allowed. This permits us to describe the problem using a set of linear constraints necessary for IP. In particular, we show that this problem can be defined as a special case of the well-known problem of set-covering (SCP), modified with additional constraints. Due to the difficulty of the problem, we have yet to discover a solution. However, we assess the potential practicality of our method by running it on smaller similar problems.

## 1. INTRODUCTION

Milton Babbitt (1916–2011) was a composer of twelve-tone serial music noted for developing complex and highly constrained music. The structures of many of his pieces are governed by a structure known as the *all-partition array*, which consists of a rectangular array of pitch-class integers, partitioned into regions of distinct "shapes", each corresponding to a distinct integer partition of 12. This structure helped Babbitt to achieve *maximal diversity* in his works, that is, the presentation of as many musical parameters in as many different variants as possible [13].

In this paper, we formalize the problem of generating an all-partition array using an integer programming paradigm in which a solution requires solving a special case of the set-covering problem (SCP), where the subsets in the cover are allowed a restricted number of overlaps with one another and where the ways in which these overlaps can occur is constrained. It turns out that this is a hard combinatorial problem. That this problem was solved by Babbitt and one of his students, David Smalley, without the use of a computer is therefore interesting in itself. Moreover, it suggests that there exists an effective procedure for solving the problem.

Construction of an all-partition array begins with an $I \times J$ matrix, $A$, of pitch-classes, $0, 1, \ldots, 11$, where each row contains $J/12$ twelve-tone rows. In this paper, we only consider matrices where $I = 6$ and $J = 96$, as matrices of this size figure prominently in Babbitt's music [13]. This results in a $6 \times 96$ matrix of pitch classes, containing $48$ twelve-tone rows. In other words, $A$ will contain an approximately uniform distribution of $48$ occurrences of each of the integers from $0$ to $11$. On the musical surface, rows of this matrix become expressed as 'musical voices', typically distinguished from one another by instrumental register [13]. A complete all-partition array is a matrix, $A$, partitioned into $K$ regions, each of which must contain each of the 12 pitch classes exactly once. Moreover, each of these regions must have a distinct "shape", determined by a distinct *integer partition* of 12 (e.g., $2+2+2+3+3$ or $1+2+3+1+2+3$) that contains $I$ or fewer summands greater than zero [7]. We denote an integer partition of an integer, $L$, by $\mathrm{IntPart}_L(s_1, s_2, \ldots, s_I)$ and define it to be an ordered set of non-negative integers, $\langle s_1, s_2, \ldots, s_I \rangle$, where $L = \sum_{i=1}^{I} s_i$ and $s_1 \geq s_2 \geq \cdots \geq s_I$. For example, possible integer partitions of 12 when $I = 6$, include $\mathrm{IntPart}_{12}(3, 3, 2, 2, 1, 1)$ and $\mathrm{IntPart}_{12}(3, 3, 3, 3, 0, 0)$. We define an *integer composition* of a positive integer, $L$, denoted by $\mathrm{IntComp}_L(s_1, s_2, \ldots, s_I)$, to also be an ordered set of $I$ non-negative integers, $\langle s_1, s_2, \ldots, s_I \rangle$, where $L = \sum_{i=1}^{I} s_i$, however, unlike an integer partition, the summands are not constrained to being in descending order of size. For example, if $L = 12$ and $I = 6$, then $\mathrm{IntComp}_{12}(3, 3, 3, 3, 0, 0)$ and $\mathrm{IntComp}_{12}(3, 0, 3, 3, 3, 0)$ are two distinct integer compositions of 12 defining the same integer *partition*, namely $\mathrm{IntPart}_{12}(3, 3, 3, 3, 0, 0)$.

Figure 1 shows a $6 \times 12$ excerpt from a $6 \times 96$ pitch-class matrix, $A$, and a region determined by the integer composition, $\mathrm{IntComp}_{12}(3, 2, 1, 3, 1, 2)$, containing each possible pitch class exactly once. Note, in Figure 1, that each summand (from left to right) in $\mathrm{IntComp}_{12}(3, 2, 1, 3, 1, 2)$, gives the number of elements in the corresponding row of the matrix (from top to bottom) in the region determined by the integer composition. We call this part of a region

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 4 | 3 | 5 | 9 | 10 | 1 | 8 | 2 | 0 | 7 | 6 |
| 6 | 7 | 0 | 2 | 8 | 1 | 10 | 9 | 5 | 3 | 4 | 11 |
| 5 | 6 | 11 | 1 | 7 | 0 | 9 | 8 | 4 | 2 | 3 | 10 |
| 2 | 9 | 10 | 8 | 4 | 3 | 0 | 5 | 11 | 1 | 6 | 7 |
| 0 | 5 | 4 | 6 | 10 | 11 | 2 | 9 | 3 | 1 | 8 | 7 |
| 1 | 8 | 9 | 7 | 3 | 2 | 11 | 4 | 10 | 0 | 5 | 6 |

**Figure 1**: A $6 \times 12$ excerpt from a $6 \times 96$ pitch-class matrix with the integer composition, $\mathrm{IntComp}_{12}(3, 2, 1, 3, 1, 2)$ (in dark gray), containing each pitch class exactly once.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 4 | 3 | **3** | 5 | 9 | 10 | 1 | 8 | 2 | 0 | 7 | 6 |
| 6 | 7 | **7** | 0 | 2 | 8 | 1 | 10 | 9 | 5 | 3 | 4 | 11 |
| 5 | 6 | 11 | 1 | 7 | 0 | 9 | 8 | 4 | 2 | 3 | 10 | |
| 2 | 9 | 10 | **10** | 8 | 4 | 3 | 0 | 5 | 11 | 1 | 6 | 7 |
| 0 | 5 | 4 | 6 | 10 | 11 | 2 | 9 | 3 | 1 | 8 | 7 | |
| 1 | 8 | 9 | 7 | 3 | 2 | 11 | 4 | 10 | 0 | 5 | 6 | |

**Figure 2**: A $6 \times 12$ excerpt from a $6 \times 96$ pitch-class matrix with a region whose shape is determined by the integer composition, $\mathrm{IntComp}_{12}(3, 3, 3, 3, 0, 0)$ (in light gray), where three elements (in bold) are horizontal insertions of pitch classes from the previous integer partition region. Note that the two indicated regions represent distinct integer partitions.

in a given row of the matrix a *summand segment*. For example, in Figure 1, the summand segment in the first row for the indicated integer partition region contains the pitch classes 11, 4 and 3. On the musical surface, the distinct shape of each integer composition helps contribute to a progression of 'musical voices' that vary in textural density, allowing for relatively thick textures in, for example, $\mathrm{IntComp}_{12}(2, 2, 2, 2, 2, 2)$ (with six participating parts) and comparatively sparse textures in, for example, $\mathrm{IntComp}_{12}(11, 0, 1, 0, 0, 0)$ (with two participating parts).

There exist a total of 58 distinct integer partitions of 12 into 6 or fewer non-zero summands [13]. An all-partition array with six rows will thus contain $K = 58$ regions, each containing every pitch class exactly once and each with a distinct shape determined by an integer composition representing a distinct integer partition. However, the number of pitch-class integers required to satisfy this constraint, $58 \times 12 = 696$, exceeds the size of a $6 \times 96$ matrix containing 576 elements, by 120. In order to satisfy this constraint, additional pitch-classes therefore have to be inserted into the matrix, with the added constraint that only horizontal insertions of at most one pitch class in each row are allowed for each of the 58 integer partition regions. Each inserted pitch class is identical to its immediate neighbor to the left, this being the right-most element of a summand segment belonging to a previous integer partition region. This constraint ensures that the order of pitch classes in the twelve-tone rows of a given row of $A$ is not altered [13]. Figure 2 shows a second integer partition region, $\mathrm{IntComp}_{12}(3, 3, 3, 3, 0, 0)$, in the matrix shown in Figure 1 (indicated in light gray), where three of its elements result from horizontal insertions of pitch classes from the previous integer partition region. Note, in Figure 2, the three horizontal insertions of pitch-class integers, 3 (in row 1), 7 (in row 2), and 10 (in row 4), required to have each pitch class occur exactly once in the second integer partition region. Not all of the 58 integer partitions must contain one or more of these insertions, however, the total number of insertions must equal the 120 additional pitch classes required to satisfy the constraint that all 58 integer partitions are represented. Note that, in order for each of the resulting integer partition regions to contain every pitch class exactly once, ten occurrences of each of the 12 pitch classes must be inserted into the matrix. This typically results in the resulting matrix being irregular (i.e., "ragged" along its right side).

In this paper, we address the problem of generating an all-partition array by formulating it as a set of linear constraints using the integer programming (IP) paradigm. In section 2, we review previous work on general IP problems and their use in the generation of musical structures. We also review previous work on the problem of generating all-partition arrays. In section 3, we introduce a way of viewing insertions of elements into the all-partition array as fixed locations in which overlaps occur between contiguous integer partition regions. In this way, our matrix remains regular and we can define the problem as a special case of the well-known IP problem of set-covering (SCP), modified so that certain overlaps are allowed between the subsets. In sections 4 and 5, we present our IP formulation of this problem as a set of linear constraints. Due to the difficulty of the problem, we have yet to discover a solution using our formulation. Nevertheless, in section 6, we present the results of using our implementation to find solutions to smaller versions of the problem and in this way explore the practicality of our proposed method. We conclude in section 7 by mentioning possible extensions to our formulation that could potentially allow it to solve the complete all-partition array generation problem.

## 2. PREVIOUS WORK

Babbitt himself laid the foundations for the construction of what would become the all-partition array during the 1960s, and he would continue to use the structure in nearly all of his later works [1–4]. Subsequent composers made use of the all-partition array in their own music and further developed ways in which its structure could be formed and used [5, 6, 11, 12, 14, 15, 17, 18, 21]. Most of these methods focus on the organization of pitch classes in a twelve-tone row and how their arrangement can make the construction of an all-partition array more likely. We propose here a more general purpose solution that will take any matrix and attempt to generate a successful structure. Furthermore, many of these previous methods were music-theoretical in nature and not explicitly computational. Work by Bazelow and Brickle is one notable exception [5, 6]. We agree here with their assessment that 'partition problems in twelve-tone theory properly belong to the study of combinatorial algorithms' [6]. However, we differ considerably in our approach and how we conceive of the structure of the all-partition array.

More recent efforts to automatically analyze and generate all-partition arrays have been based on backtracking algorithms. [7–9]. True to the structure of the all-partition array (as it appears on the musical surface) and the way in which Babbitt and other music theorists conceive of its structure, these attempts to generate an all-partition array form regions of pitch classes according to the process described in section 1, where horizontal repetitions of pitch-classes are added, resulting in an irregular matrix. While these existing methods have further proposed various heuristics to limit the solution space or allow for incomplete solutions, they were unable to generate a complete all-partition array [7–9].

In general, for difficult combinatorial problems, more efficient solving strategies than backtracking exist. One such example is integer programming (IP). IP is a computationally efficient and practical paradigm for dealing with typically NP-hard problems, such as the traveling salesman, set-covering and set-partitioning problems, where these are expressed using only linear constraints (i.e., equations and inequalities) and a linear objective function [10, 16]. One benefit of using IP, is that it allows for the separation of the formulation of a problem by users and the development by specialists of an algorithm for solving it. Many of these powerful solvers dedicated to IP problems have been developed and used particularly in the field of operations research. Compared to approximate computational strategies, such as genetic algorithms, IP formulations and their solvers are suitable for searching for solutions that strictly satisfy necessary constraints. For this reason, we expect that the IP paradigm could provide an appropriate method for approaching the problem of generating all-partition arrays.

In recent work, IP has been applied to problems of analysis and generation of music [19, 20]. This is of importance to the research presented here as it demonstrates the relevance of these traditional optimization problems of set-covering (SCP) and set-partitioning (SPP), to general problems found in computational musicology, where SPP has been used in the segmentation of melodic motifs and IP has been used in describing global form. In the next section, we address the set-covering problem (SCP) in greater detail and show how it is related to the problem of generating all-partition arrays.

## 3. SET-COVERING PROBLEM FORMULATION OF ALL-PARTITION ARRAY GENERATION

The set-covering (SCP) problem is a well-known problem in computer science and operations research that can be shown to be NP-hard [10]. Let $E$ be a set whose elements are $\{E_1, E_2, \cdots, E_{\#E}\}$ (where $\#E$ denotes the number of elements in $E$), $F$ be a family of subsets of $E$, $\{F_1, F_2, \cdots, F_{\#F}\}$, and $S$ be a subset of $F$. By assigning a constant cost, $c_s$, to each $F_s$, the objective of the set-covering problem (SCP) is to

$$\underset{S \subset F}{Minimize} \sum_{F_s \in S} c_s$$
$$subject\ to \bigcup_{F_s \in S} F_s = E.$$

In other words, a solution $S$ is a *cover* of $E$ that allows for the same elements to appear in more than one subset, $F_s$. In this section, we suggest that our problem can be viewed as an SCP with additional constraints.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 4 | 3 | 5 | 9 | 10 | 1 | 8 | 2 | 0 | 7 | 6 |
| 6 | 7 | 0 | 2 | 8 | 1 | 10 | 9 | 5 | 3 | 4 | 11 |
| 5 | 6 | 11 | 1 | 7 | 0 | 9 | 8 | 4 | 2 | 3 | 10 |
| 2 | 9 | 10 | 8 | 4 | 3 | 0 | 5 | 11 | 1 | 6 | 7 |
| 0 | 5 | 4 | 6 | 10 | 11 | 2 | 9 | 3 | 1 | 8 | 7 |
| 1 | 8 | 9 | 7 | 3 | 2 | 11 | 4 | 10 | 0 | 5 | 6 |

**Figure 3**: A $6 \times 12$ excerpt from a $6 \times 96$ pitch-class matrix with two integer compositions, $\mathrm{IntComp}_{12}(3, 2, 1, 3, 1, 2)$ (in dark gray and outlined) and $\mathrm{IntComp}_{12}(3, 3, 3, 3, 0, 0)$ (in light gray), that form distinct integer partition regions. Note, that the second composition overlaps three fixed locations in the first.

### 3.1 All-partition array generation as a set-covering problem (SCP) with additional constraints

When viewing the all-partition array in the context of $\bigcup_{F_s \in S} F_s = E$ above, $E$ is the set that consists of all locations $(i, j)$ in the matrix, $A$, and $F_s$ are the sets of locations $(i, j)$ that correspond to the "shapes" of integer compositions. We call each $F_s$ a *candidate set*. A candidate set $F_s$ is characterized by two conditions that we call *containment* and *consecutiveness*. Containment means that the elements (i.e., locations $(i, j)$) of $F_s$ correspond to twelve distinct integers, $0, 1, \ldots, 11$, in $A$. Consecutiveness means that each of its elements belonging to the same row in $A$ are consecutive. In this sense, $F$ includes all sets found in $A$ that satisfy the conditions of consecutiveness and containment.

As the expression $\bigcup_{F_s \in S} F_s = E$ implies, a candidate set is allowed to share elements with another candidate set. Similarly, the pitch classes in $A$ (i.e., corresponding to elements in $E$) that become insertions in the original problem can be instead regarded as shared elements or *overlaps* between contiguous integer composition regions, with the result that the matrix remains regular. Figure 3 shows how these overlaps would occur in the two integer composition regions shown in Figure 2.

Viewed in this way, a solution to the problem of generating an all-partition array thus satisfies the basic criterion of an SCP, namely, the condition for *set-covering*, $\bigcup_{F_s \in S} F_s = E$. However, this criterion alone fails to account for the unique constraints under which such a covering is formed in an all-partition array. In the original SCP, there are no constraints on the order of subsets, the order of their elements or the number of overlaps and the ways in

which they can occur. On the other hand, an all-partition array must satisfy such additional conditions. We denote the constraints for satisfying such additional conditions by $Add.\ Conditions$.

$Add.\ Conditions$ includes the conditions in the all-partition array governing (1) the left-to-right order of contiguous candidate sets, (2) permissible overlaps between such sets, and (3) the *distinctness* of sets in $S$. This last condition of distinctness ensures that the integer compositions used in a cover, $S$, define every possible integer partition once and only once. On the other hand, the conditions for set-covering, $\bigcup_{F_s \in S} F_s = E$, are conditions of (1) candidate sets (which satisfy containment and consecutiveness) and (2) *covering*, meaning that each element in $E$ is covered no less than once.

We can now state that our problem of generating an all-partition array is to

$$\begin{aligned} &Minimize \underset{S \subset F}{\sum_{F_s \in S}} c_s \\ &subject\ to \bigcup_{F_s \in S} F_s = E, \\ &\qquad Add.\ Conditions. \end{aligned}$$

where the associated cost, $c_s$, of each $F_s$, can be interpreted as a preference for one integer composition or another. It is likely that, in the interest of musical expression, Babbitt may have preferred the shapes of some integer partition regions over others [13]. However, as his preference is unknown, we can regard these costs to have the same value for each $F_s$.

Due to the condition of distinctness (just described), $|S|$ can be fixed at 58. This feature, combined with the equal costs of each $F_s$, means that the objective function, $\sum_{F_s \in S} c_s$, for this problem, is constant. For these reasons, the above formulation is a *constraint satisfaction problem*. This motivates our discussions in sections 6 and 7 on possible alternative objective functions.

In the next two sections, we implement the constraint satisfaction problem defined above using integer programming (IP). In particular, section 4 addresses the conditions for set-covering, $\bigcup_{F_s \in S} F_s = E$, and section 5 addresses those in $Add.\ Conditions$. It is because of our new way of viewing this problem, with a regular matrix and overlaps, that we are able to introduce variables for use in IP to describe these conditions.

## 4. IP IMPLEMENTATION OF CONDITIONS FOR SET-COVERING IN ALL-PARTITION ARRAY GENERATION

In this section, we introduce our set of linear constraints for satisfying the general conditions for set-covering, $\bigcup_{F_s \in S} F_s = E$, in the generation of an all-partition array. Before we introduce these constraints, we define the necessary variables and constants used in our implementation of the conditions for set-covering. We begin with a given matrix found in one of Babbitt's works based on the all-partition array. Examples of the matrices used in this paper can be found in Babbitt's *Arie da Capo* (1974) and *None but the Lonely Flute* (1991), among others. Let $(A_{i,j})$ be a $(6, 96)$-matrix whose elements are the pitch-class integers, $0, 1, \ldots, 11$. We denote the number of rows and columns by $I$ and $J$, respectively.

Let $x_{i,j,k}$ $(1 \leq i \leq I,\ 1 \leq j \leq J)$ be a binary variable corresponding to each location $(i, j)$ in $A$ and a subset (i.e., integer partition) identified by the integer $k$, where $1 \leq k \leq K$ and $K = 58$. Here, we consider the case where $I = 6$ and $J = 96$, so there are 58 sets of 576 such variables. Each of these variables will indicate whether or not a location $(i, j)$ belongs to a candidate set for the $k$th position in the sequence of 58 integer partition regions. We denote the set of locations $(i, j)$ whose corresponding value for $x_{i,j,k}$ is 1, to be $C_k$. Subject to conditions for consecutiveness and containment, $C_k$ will be a candidate set.

Let $(B^p_{i,j})$ $(0 \leq p \leq 11)$ be constant matrices, equal in size to $A$, where $B^p_{i,j} = 1$ if and only if $A_{i,j} = p$ and $B^p_{i,j} = 0$ otherwise. The locations $(i, j)$ whose values of $B^p_{i,j}$ equal 1, correspond to the locations of pitch-class $p$ in $A$.

### 4.1 Conditions for $C_k$ to contain twelve distinct integers in $A$ (condition of containment)

A condition for $C_k$ to satisfy the condition of containment is that its number of elements is 12 and each corresponds to a distinct pitch-class in $A$. These conditions are expressed by the following two equations:

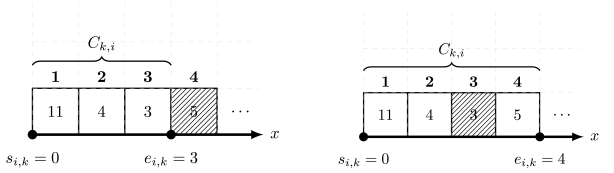
$$\forall k \in [1, K],\ \sum_{i=1}^{I} \sum_{j=1}^{J} x_{i,j,k} = 12, \tag{1}$$

$$\forall p \in [0, 11], \forall k \in [1, K],\ \sum_{i=1}^{I} \sum_{j=1}^{J} B^p_{i,j} \cdot x_{i,j,k} = 1. \tag{2}$$

Because $x_{i,j,k}$ equals 1 if $(i, j)$ is included in $C_k$ and 0 if it is not, Equation 1 means that there are 12 elements in $C_k$. In Equation 2, we ensure that each corresponding pitch-class integer $p$ for the elements in $C_k$, appears once and only once.

### 4.2 Conditions for $C_k$ to be integer compositions in $A$ (condition of consecutiveness)

Let $C_{k,i}$ be the $i$th-row part of $C_k$ (i.e., the summand segment of composition $k$ for row $i$). Let $s_{i,k}$ be an integer variable corresponding to the x-coordinate of a 'starting point', which lies at the left side of the leftmost component of $C_{k,i}$. The value of $s_{i,k}$ is then equal to the column number of the leftmost component of $C_{k,i}$, minus 1. The origin point of this coordinate lies along the left hand side of the matrix $A$, and we set the width of each location $(i, j)$ to be 1. Similarly, let $e_{i,k}$ be an integer variable corresponding to the x-coordinate of an 'ending point', which lies at the right side of the rightmost component belonging to $C_{k,i}$. The value of $e_{i,k}$ is then equal to the column number of the




(a) $C_{k,i}$ contains pitch classes $11, 4, 3$ $(j = 1, 2, 3)$ and satisfies consecutiveness.

(b) $C_{k,i}$ contains pitch classes $11, 4, 5$ $(j = 1, 2, 4)$ and does not satisfy consecutiveness.

**Figure 4**: Two $C_{k,i}$ and corresponding $s_{i,k}$ and $e_{i,k}$ from Figure 3 when $k = 1$ and $i = 1$. Shaded elements indicate $x_{i,j,k} = 0$ and unshaded elements indicate $x_{i,j,k} = 1$.

rightmost component of $C_{k,i}$. Figure 4 shows an example of two possible $C_{k,i}$ from Figure 3. If there is no component in $C_{k,i}$ $(k \geq 2)$, we define $s_{i,k}$ to be $e_{i,k-1}$ and $e_{i,k}$ to be $s_{i,k}$. If there is no component in $C_{1,i}$, we define $s_{i,k}$ and $e_{i,k}$ to be 0. Then, $s_{i,k}$ and $e_{i,k}$ are subject to the following constraint of range:

$$\forall i \in [1, I], \forall k \in [1, K],\ 0 \leq s_{i,k} \leq e_{i,k} \leq J. \quad (3)$$

The condition under which $C_k$ $(k \in [1, K])$ forms an integer composition—that is, satisfies the condition of consecutiveness, is expressed by the following three constraints:

$$\forall i \in [1, I], \forall j \in [1, J], \forall k \in [1, K], \quad j \cdot x_{i,j,k} \leq e_{i,k}, \quad (4)$$

$$\forall i \in [1, I], \forall j \in [1, J], \forall k \in [1, K], \quad J - s_{i,k} \geq (J + 1 - j) \cdot x_{i,j,k}, \quad (5)$$

$$\forall i \in [1, I], \forall k \in [1, K],\ \sum_{j=1}^{J} x_{i,j,k} = e_{i,k} - s_{i,k}. \quad (6)$$

In Equation 4, each element of $C_{k,i}$ must be located at column $e_{i,k}$ or to the left of column $e_{i,k}$. Equation 5 states that each element of $C_{k,i}$ must be located at column $s_{i,k} + 1$ or to the right of column $s_{i,k} + 1$. Equation 6, combined with the previous two constraints, states that the length of $C_{k,i}$ must be equal to $e_{i,k} - s_{i,k}$, implying that the column numbers $j$ of the elements in $C_{k,i}$ are consecutive from $s_{i,k} + 1$ to $e_{i,k}$, where $C_{k,i}$ contains at least one element.

### 4.3 Condition for covering $A$

As every location $(i, j)$ in $A$ (i.e., $E$ in our SCP) must be covered at least once, we pose the following condition of covering:

$$\forall i \in [1, I], \forall j \in [1, J],\ \sum_{k=1}^{K} x_{i,j,k} \geq 1. \quad (7)$$

Equation 1 states that for all $K = 58$ integer partitions, there are $12 \cdot K = 696$ variables, $x_{i,j,k}$, that will equal 1. A successful cover of $A$ by Equation 7, however, states that all of $I \cdot J = 576$ places $(i, j)$ in $A$, are covered once or more than once. Collectively, these imply that there are 120 or less than 120 places (i.e., combinations of $(i, j)$) that are covered twice or more than twice. These 120 overlaps correspond to the 120 insertions of pitch-class integers used when constructing an all-partition array in its original form. By satisfying all of the constraints above, each $C_k$ forms a candidate set (i.e., a member of $F$ in our SCP) and the condition for set-covering, $\bigcup_{F_s \in S} F_s = E$, is satisfied.

## 5. IP IMPLEMENTATION OF ADDITIONAL CONDITIONS IN ALL-PARTITION ARRAY GENERATION

In this section, we introduce our set of additional linear constraints beyond those required for satisfying the condition of set-covering in the SCP.

### 5.1 Left-to-right order of $C_k$ and permissible overlaps

$C_k$ must be located immediately to the right of $C_{k-1}$. This is expressed by

$$\forall i \in [1, I], \forall k \in [2, K],\ e_{i,k-1} \leq e_{i,k}, \quad (8)$$

$C_{k-1,i}$ and $C_{k,i}$ may overlap by no more than one element. This is expressed by the following inequality:

$$\forall i \in [1, I], \forall k \in [2, K],\ e_{i,k-1} - 1 \leq s_{i,k} \leq e_{i,k-1}, \quad (9)$$

meaning that $s_{i,k}$ will be equal to $e_{i,k-1}$ if there is no overlap and $s_{i,k}$ will be equal to $e_{i,k-1} - 1$ if there is an overlap.

### 5.2 Conditions for $C_k$ to be integer compositions defining distinct integer partitions (condition of distinctness)

Let $y_{i,k,l}$ be a binary variable that indicates whether or not the length of $C_{k,i}$ is greater than or equal to $l$ $(1 \leq l \leq L, L = 12)$, by introducing the following constraints:

$$\forall i \in [1, I], \forall k \in [1, K],\ e_{i,k} - s_{i,k} = \sum_{l=1}^{L} y_{i,k,l}, \quad (10)$$

$$\forall i \in [1, I], \forall k \in [1, K], \forall l \in [2, L], \quad y_{i,k,l-1} \geq y_{i,k,l}. \quad (11)$$

Equation 10 states that the sum of all elements in $\langle y_{i,k,1}, y_{i,k,2}, \ldots, y_{i,k,L} \rangle$ is equal to the length of $C_{k,i}$, while Equation 11 states that its elements equal to 1 begin in the first position and are consecutive (e.g., $\langle 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$, when the length of $C_{k,i}$ is 3.)

The number of the lengths of $C_{k,i}$ $(1 \leq i \leq I)$ that are greater than or equal to $l$ is given by $\sum_{i=1}^{I} y_{i,k,l}$. The twelve values of $\sum_{i=1}^{I} y_{i,k,l}$ $(1 \leq l \leq L)$ then, will precisely represent the type of partition. For example, if $C_k$ is $\mathrm{IntComp}_{12}(3, 2, 1, 3, 1, 2)$, then $y_{i,k,l}$ $(\forall i \in [1, I], \forall l \in [1, L])$ would be

1,1,1,0,0,0,0,0,0,0,0,0
1,1,0,0,0,0,0,0,0,0,0,0
1,0,0,0,0,0,0,0,0,0,0,0
1,1,1,0,0,0,0,0,0,0,0,0
1,0,0,0,0,0,0,0,0,0,0,0
1,1,0,0,0,0,0,0,0,0,0,0

and $\sum_{i=1}^{I} y_{i,k,l}$ would be $[6, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]$.

We denote the number of all integer partitions by $N$ ($N = K = 58$) and denote a single integer partition $n$ ($1 \leq n \leq N$) by $P_n$. We can express $P_n$ as $[P_{n,1}, P_{n,2}, \ldots, P_{n,L}]$ ($1 \leq n \leq N$), where $P_{n,l}$ corresponds to the twelve values $\sum_{i=1}^{I} y_{i,k,l}$ ($1 \leq l \leq L$) described above.

Then, by implementing the following expression:

$$\forall k \in [1, K], \forall n \in [1, N], \forall l \in [1, L], \quad P_{n,l} \cdot z_{k,n} \leq \sum_{i=1}^{I} y_{i,k,l}. \tag{12}$$

we can express whether $C_k$ defines the integer partition $n$ or not by the binary variable $z_{k,n}$. For example, if $z_{k,n} = 0$, the value of $P_{n,l} \cdot z_{k,n} = 0$ constrains nothing, and thus $C_k$ cannot be the integer partition $n$ (because of the next equation). On the other hand, if $z_{k,n} = 1$, $C_k$ must be the integer partition $n$. Accordingly, $z_{k,n}$ will equal 1 only if the twelve values $\sum_{i=1}^{I} y_{i,k,l}$ correspond to $P_n$. From this, determining whether or not all different partitions are present can be expressed by the following equation:

$$\forall n \in [1, N], \ \sum_{k=1}^{K} z_{k,n} = 1. \tag{13}$$

## 6. EXPERIMENTS

In order to determine whether or not our formulation works as intended, we implemented the constraints described in sections 4 and 5 and supplied these to an IP solver based on branch-and-bound (Gurobi Optimizer). As the objective function in our formulation amounts to a constant-cost function (described in section 3), we replaced it with a non-constant objective function, $\sum_{i,j,k} c_{i,j,k} \cdot x_{i,j,k}$, where $c_{i,j,k}$ assumes a randomly generated integer for promoting this process of branch and bound. When the first feasible solution is found, we stop the search.

Although we first attempted to find a complete all-partition array, we were unable to discover a solution after one day of calculation. This highlights the difficulty of the problem and reinforces those findings by previous methods that were similarly unable to find a complete all-partition array [7]. As the target of our current formulation is only solutions which strictly satisfy all constraints, we opted to try finding complete solutions to smaller-sized problems, using the first $j$ columns of the original matrix. Because we cannot use all 58 integer partitions in the case $K < N$, a slight modification to Equation 13 was needed for this change. Its equality was replaced by $\leq$ and an additional constraint, $\forall k \in [1, K], \ \sum_{n=1}^{N} z_{k,n} = 1$, for allocating one partition to each $C_k$, was added.

Figure 5 shows the duration (vertical axis) of time spent on finding a solution in matrices of varying size. The number of integer compositions, $K$, was set to $(J+2)/2$, where $J$ is an even number. This ensures that a given solution will always contain 12 overlaps. These findings suggest that the necessary computational time in finding a solution tends to dramatically increase with an increase in $J$. However, this increase fluctuates, suggesting that each small matrix represents a unique problem space with different sets of difficulties (e.g., the case $J = 14$ was unfeasible). For this reason, finding a solution in a complete matrix (6,96) within a realistic limitation of time would be difficult for our current method, even using a fast IP solver. This strongly motivates future improvements as well as the possibility of an altogether different strategy.

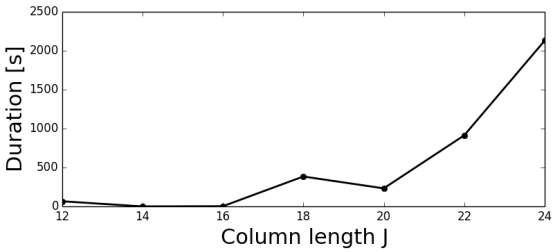



**Figure 5**: Duration of time spent on finding the first solution for each small matrix, whose column length is $J (12 \leq J \leq 24, J \in 2\mathbb{N})$. $K$ is set to $(J + 2)/2$, resulting in 12 overlaps. Note, that no feasible solution exists when $J = 14$.

## 7. CONCLUSION AND FUTURE WORK

In this paper, we have introduced a novel integer-programming-based perspective on the problem of generating Milton Babbitt's all-partition arrays. We have shown that insertions and the irregular matrix that results can be replaced with restricted overlaps, leaving the regular matrix unchanged. This view allows us to formulate the problem as a set-covering problem (SCP) with additional constraints and then implement it using integer programming. Due to the difficulty of the problem, we have so far been unable to find a solution. However, we have been able to produce solutions in a practical running time ($< 2500$ seconds) when the matrix is reduced in size to 24 columns or less. These results motivate possible extensions to our formulation. First, a relaxation of the problem is possible, for example, by using an objective function that measures the degree of incompleteness of a solution. This could allow for approximate solutions to be discovered, such as those found in previous work [7]. Second, it may be the case that a solution to the full problem may be achievable by combining solutions to smaller subproblems that we have shown to be solvable in a practical running time.

## 8. ACKNOWLEDGEMENTS

The work of Tsubasa Tanaka reported in this paper was supported by JSPS Postdoctoral Fellowships for Research Abroad. The work of Brian Bemman and David Meredith was carried out as part of the project Lrn2Cre8, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 610859.

## 9. REFERENCES


[1] Milton Babbitt. Twelve-tone invariants as compositional determinants. *Journal of Music Theory*, 46(2):246–259, 1960.

[2] Milton Babbitt. Set structure as a compositional determinant. *Journal of Music Theory*, 5(1):72–94, 1961.

[3] Milton Babbitt. Twelve-tone rhythmic structure and the electronic medium. *Perspectives of New Music*, 1(1):49–79, 1962.

[4] Milton Babbitt. Since Schoenberg. *Perspectives of New Music*, 12(1/2):3–28, 1973.

[5] Alexander R. Bazelow and Frank Brickle. A partition problem posed by Milton Babbitt (Part I). *Perspectives of New Music*, 14(2):280–293, 1976.

[6] Alexander R. Bazelow and Frank Brickle. A combinatorial problem in music theory: Babbitt's partition problem (I). *Annals of the New York Academy of Sciences*, 319(1):47–63, 1979.

[7] Brian Bemman and David Meredith. Comparison of heuristics for generating all-partition arrays in the style of Milton Babbitt. In *CMMR 11th International Symposium on Computer Music Multidisciplinary Research, 16–19 June 2015*, Plymouth, UK, 2015.

[8] Brian Bemman and David Meredith. Exact cover problem in Milton Babbitt's all-partition array. In Tom Collins, David Meredith, and Anja Volk, editors, *Mathematics and Computation in Music: Fifth International Conference, MCM 2015, London, UK, June 22–25, 2015, Proceedings*, volume 9110 of *Lecture Notes in Artificial Intelligence*, pages 237–242. Springer, Berlin, 2015.

[9] Brian Bemman and David Meredith. Generating Milton Babbitt's all-partition arrays. *Journal of New Music Research*, 45(2), 2016. `http://www.tandfonline.com/doi/full/10.1080/09298215.2016.1172646`.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3rd edition, 2009.

[11] David Kowalski. *The array as a compositional unit: A study of derivational counterpoint as a means of creating hierarchical structures in twelve-tone music*. PhD thesis, Princeton University, 1985.

[12] David Kowalski. The construction and use of self-deriving arrays. *Perspectives of New Music*, 25(1), 1987.

[13] Andrew Mead. *An Introduction to the Music of Milton Babbitt*. Princeton University Press, Princeton, NJ., 1994.

[14] Robert Morris. *Composition with Pitch-Classes: A Theory of Compositional Design*. Yale University Press, New Haven, CT, 1987.

[15] Robert Morris. *The Whistling Blackbird: Essays and Talks on New Music*. The University of Rochester Press, Princeton, NJ, 2010.

[16] A. J. Orman and H. Paul Williams. A survey of different integer programming formulations of the travelling salesman problem. In Erricos John Kontoghiorghes and Cristian Gatu, editors, *Optimisation, Econometric and Financial Analysis*, volume 9 of *Advances in computational management science*, pages 93–108. Springer-Verlag Berlin Heidelberg, Berlin, 2006.

[17] Daniel Starr and Robert Morris. A general theory of combinatoriality and the aggregate, part 1. *Perspectives of New Music*, 16(1):3–35, 1977.

[18] Daniel Starr and Robert Morris. A general theory of combinatoriality and the aggregate, part 2. *Perspectives of New Music*, 16(2):50–84, 1978.

[19] Tsubasa Tanaka and Koichi Fujii. Describing global musical structures by integer programming. In Tom Collins, David Meredith, and Anja Volk, editors, *Mathematics and Computation in Music: Fifth International Conference, MCM 2015, London, UK, June 22–25, 2015, Proceedings*, volume 9110 of *Lecture Notes in Artificial Intelligence*, pages 52–63. Springer, Berlin, 2015.

[20] Tsubasa Tanaka and Koichi Fujii. Melodic pattern segmentation of polyphonic music as a set partitioning problem. In *International Congress on Music and Mathematics, Puerto Vallarta, November 26–29, 2014, Proceedings*. Springer, Berlin, to be published.

[21] Godfrey Winham. Composition with arrays. *Perspectives of New Music*, 9(1):43–67, 1970.

# Constraint programming approach to the problem of generating Milton Babbitt's all-partition arrays

Tsubasa Tanaka[1], Brian Bemman[2] and David Meredith[2]

[1] STMS Lab : IRCAM, CNRS, UPMC, Paris, France
tsubasa.tanaka@ircam.fr
[2] Aalborg University, Aalborg, Denmark
{bb,dave}@create.aau.dk

**Abstract.** Milton Babbitt (1916–2011) was a composer of twelve-tone serial music noted for creating the *all-partition array*. One part of the problem in generating an all-partition array requires finding a covering of a pitch-class matrix by a collection of sets, each forming a region containing 12 distinct elements and corresponding to a distinct integer partition of 12. Constraint programming (CP) is a tool for solving such combinatorial and constraint satisfaction problems. In this paper, we use CP for the first time to formalize this problem in generating an all-partition array. Solving the whole of this problem is difficult and few known solutions exist. Therefore, we propose solving two sub-problems and joining these to form a complete solution. We conclude by presenting a solution found using this method. Our solution is the first we are aware of to be discovered automatically using a computer and differs from those found by composers.

**Keywords:** Babbitt · all-partition array · computational musicology · constraint programming

## 1 Introduction

Milton Babbitt (1916–2011) was a composer of twelve-tone serial music noted for developing highly constrained and often complex musical structures. Many of his pieces are organized according to one such structure known as the *all-partition array* [1]. An all-partition array is a covering of a matrix of pitch-class integers by a collection of sets, each of which forms a region in this matrix containing 12 distinct pitch classes from consecutive elements in its rows and that corresponds to a distinct integer partition of 12 (to be clarified in the next section). This unique structure imposes a strict organization on the pitch classes in his works, and it serves as both a method of musical composition and musical form. Moreover, the all-partition array allowed Babbitt one of many ways to achieve *maximal diversity* in his music.[3]

[3] Maximal diversity is the presentation of as many musical parameters in as many different ways as possible [2].

In this paper, we formulate one part of the problem in generating an all-partition array, beginning from a given matrix of pitch-class integers, using constraint programming (CP) and with a particular focus on its mathematical aspects. Using our model and a method for dividing this matrix into smaller, sub-problems, we obtained a solution, which, we believe, is the first to be discovered automatically using a computer and differs from those found by composers. CP is a programming paradigm that has been successfully applied to the solving of various constraint satisfaction problems in music [3–7]. It seems natural then, that CP could be used in the problem we address here. Moreover, having such a model could, for example, be used as a basis for generating new musical works.

### 1.1 The structure of an all-partition array

In this section, we describe the structure of an all-partition array in a way that assumes the reader has a basic understanding of pitch class set theory. Constructing an all-partition array begins with the construction of an $I \times J$ matrix, $A$, whose elements are pitch-class integers, $0, 1, \ldots, 11$, where each row contains $J/12$ twelve-tone rows. The dimensions of this matrix constrain the most important requirement of the structure of an all-partition array, however, Babbitt generally limited himself to sizes of $4 \times 96$, $6 \times 96$, and $12 \times 72$ [2]. In this paper, we consider only matrices where $I = 4$ and $J = 96$, as matrices of this size figure prominently in Babbitt's music [2]. This results in a $4 \times 96$ matrix of pitch classes, containing 32 twelve-tone rows from the possible 48 related by any combination of transposition, inversion and retrograde (i.e., reversal). In other words, $A$ will contain an approximately uniform distribution of 32 occurrences of each of the integers from 0 to 11.[4] On the musical surface, rows of this matrix become expressed as 'musical voices', typically distinguished from one another by instrumental register [2].

A complete all-partition array is a covering of matrix, $A$, by $K$ sets, each of which is itself a partition of the set $\{0, 1, \ldots, 11\}$ whose parts (1) contain consecutive row elements from $A$ and (2) have cardinalities equal to the summands in one of the $K$ distinct integer partitions of 12 (e.g., $6 + 6$ or $5 + 4 + 2 + 1$) containing $I$ or fewer summands greater than zero.[5] Figure 1 shows a $4 \times 12$ excerpt from a $4 \times 96$ pitch-class matrix, $A$, and two such sets forming *regions* in $A$ each containing every pitch class exactly once and corresponding to two distinct integer partitions, whose exact "shapes" are more precisely represented as the *integer compositions*, $\mathrm{IntComp}_{12}(4, 4, 4, 0)$ and $\mathrm{IntComp}_{12}(0, 6, 3, 3)$.[6]

[4] For a more detailed description of the constraints governing the organization of matrices in Babbitt's music, see [2, 8].

[5] We denote an integer partition of an integer, $L$, by $\mathrm{IntPart}_L(s_1, s_2, \ldots, s_I)$ and define it to be an ordered set of non-negative integers, $\langle s_1, s_2, \ldots, s_I \rangle$, where $L = \sum_{i=1}^{I} s_i$ and $s_1 \geq s_2 \geq \cdots \geq s_I$.

[6] We define an *integer composition* of a positive integer, $L$, denoted by $\mathrm{IntComp}_L(s_1, s_2, \ldots, s_I)$, to also be an ordered set of $I$ non-negative integers, $\langle s_1, s_2, \ldots, s_I \rangle$, where $L = \sum_{i=1}^{I} s_i$.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 10 | 3 | 7 | 6 | 2 | 4 | 5 | 8 | 1 | 9 | 0 |
| 8 | 9 | 4 | 0 | 1 | 5 | 3 | 2 | 11 | 6 | 10 | 7 |
| 2 | 5 | 1 | 6 | 9 | 10 | 0 | 8 | 7 | 11 | 4 | 3 |
| 7 | 4 | 8 | 3 | 0 | 11 | 9 | 1 | 2 | 10 | 5 | 6 |

Fig. 1: A $4 \times 12$ excerpt from a $4 \times 96$ pitch-class matrix with two distinct integer partition regions represented precisely by the integer compositions, $\text{IntComp}_{12}(4, 4, 4, 0)$ (in dark gray) and $\text{IntComp}_{12}(0, 6, 3, 3)$ (in light gray), each containing every pitch class exactly once.

Note, in Figure 1, that each summand (from left to right) in $\text{IntComp}_{12}(4, 4, 4, 0)$, gives the number of elements in the corresponding row of the matrix (from top to bottom) in this region. Unlike common tiling problems using, for example, polyominoes, these regions need not have connected interiors, as demonstrated by the second region in Figure 1 between rows 3 and 4. On the musical surface, the distinct shape of each region helps contribute to a progression of 'musical voices' that vary in textural density, allowing for relatively thick textures in, e.g., $\text{IntComp}_{12}(3, 3, 3, 3)$ (with four participating parts) and comparatively sparse textures in, e.g., $\text{IntComp}_{12}(11, 0, 1, 0)$ (with two participating parts).

There exist a total of 34 distinct integer partitions of 12 into 4 or fewer non-zero summands [2]. An all-partition array with four rows will thus contain $K = 34$ regions, each containing every pitch class exactly once and each with a distinct "shape" determined by an integer composition defining a distinct integer partition. However, the number of pitch classes required to satisfy this constraint, $34 \times 12 = 408$, exceeds the size of a $4 \times 96$ matrix containing 384 elements, by 24. In order to satisfy this constraint, contiguous regions may share pitch classes, with the added constraint that only horizontal *overlaps* of at most one pitch class in each row are allowed for each of the 34 integer partition regions. Figure 2 shows a third region, $\text{IntComp}_{12}(5, 1, 0, 6)$ (in medium gray), in the matrix shown in Figure 1, where two of its elements result from overlapped pitch classes from previous regions. Note, in Figure 2, the two horizontal overlaps of pitch class, 7

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 10 | 3 | 7 | 6 | 2 | 4 | 5 | 8 | 1 | 9 | 0 |
| 8 | 9 | 4 | 0 | 1 | 5 | 3 | 2 | 11 | 6 | 10 | 7 |
| 2 | 5 | 1 | 6 | 9 | 10 | 0 | 8 | 7 | 11 | 4 | 3 |
| 7 | 4 | 8 | 3 | 0 | 11 | 9 | 1 | 2 | 10 | 5 | 6 |

Fig. 2: A $4 \times 12$ excerpt from a $4 \times 96$ pitch-class matrix with a third integer composition, $\text{IntComp}_{12}(5, 1, 0, 6)$ (in medium gray), sharing one pitch class from each of the two previous regions.

(in row 1 and belonging to the first region) and 8 (in row 4 and belonging to the second region), required to have each pitch class occur exactly once in the third

integer partition region. This means that while contiguous regions may share pitch classes, such regions need not be necessarily adjacent in sequence.

Composers have primarily relied on constructing all-partition arrays by hand and at least some of their methods have been published [1,9,10]. Algorithms for automating this task have also been proposed [8,11]. However, generating an all-partition array is a large combinatorial problem and satisfying the constraints of its structure is difficult. To date, none of these algorithms have been able to solve this problem automatically. This observation motivates our decision here to look for alternative programming paradigms and methods for possibly better addressing this problem. In section 2, we present our CP constraints for implementing the problem of generating an all-partition array from a given matrix. As solving for the entire matrix directly is difficult, in section 3, we present a method of dividing this matrix into two smaller matrices, choosing integer partitions based on how frequently they appear in solutions to one of these smaller matrices, and re-joining them to form a complete solution. We conclude here with a solution discovered using this method.

## 2 CP constraints for the problem of generating an all-partition array from a given matrix

We begin the discussion of our CP constraints for generating an all-partition array, with a given matrix found in one of Babbitt's works based on the all-partition array.[7] Let $(A_{i,j})$ be this $(4, 96)$-matrix whose elements are the pitch-class integers, $0, 1, \ldots, 11$. We denote the number of rows and columns by $I$ and $J$, respectively. Let $x_{i,j,k}$ $(1 \leq i \leq I,\ 1 \leq j \leq J)$ be a binary variable corresponding to each location $(i, j)$ in $A$ and a subset (i.e., a region) identified by the integer $k$, where $1 \leq k \leq K$ and $K = 34$. There are then 34 sets of 384 such variables. Each of these variables will indicate whether or not a location $(i, j)$ belongs to a *candidate set*, which we denote, $C_k$, for the $k$th position in the sequence of 34 regions. For $C_k$ to be a candidate set, it must form a region in $A$ (as described in section 1), by satisfying two conditions, *consecutiveness* and *containment*, which we will introduce below. Having satisfied these conditions, $C_k$ will be a candidate set in a possible solution to our problem, in which its elements correspond to 12 distinct pitch classes in $A$ and whose "shape" is defined by an integer composition. Additional constraints e.g., ensuring that each of these candidate sets is then a distinct integer partition and that their overlaps do not exceed one in each row, will then complete our formulation of this problem.

### 2.1 Consecutiveness

The condition of consecutiveness states that pitch classes belonging to the same region and row in $A$ must lie adjacent to one another with no gaps between. We

[7] Examples of this matrix can be found in Babbitt's *My Ends are My Beginnings* (1978) and *Beaten Paths* (1988), among others.

ensure this is the case by placing constraints on the strings of 0's and 1's that are allowed in the rows formed by $\langle x_{i,1,k}, x_{i,2,k}, \ldots, x_{i,J,k}\rangle$ for each $(i,k)$. If, for example, the string $\langle \ldots, 0, 1, \ldots\rangle$ appears in the $i$th row for some $k$, then there can be no 1 occurring before 0. This is expressed by the following:

$$\forall i \in [1,I], \forall j \in [3,J], \forall k \in [1,K],$$
$$(x_{i,j-1,k} = 0 \ \wedge \ x_{i,j,k} = 1) \Longrightarrow \bigwedge_{j'=1}^{j-2} (x_{i,j',k} = 0). \tag{1}$$

On the other hand, if $\langle \ldots, 1, 0, \ldots\rangle$ appears in this row, then there can be no 1 after 0. This is expressed by the following:

$$\forall i \in [1,I], \forall j \in [1,J-2], \forall k \in [1,K],$$
$$(x_{i,j,k} = 1 \ \wedge \ x_{i,j+1,k} = 0) \Longrightarrow \bigwedge_{j'=j+2}^{J} (x_{i,j',k} = 0). \tag{2}$$

In other words, all 1's in $\langle x_{i,1,k}, x_{i,2,k}, \ldots, x_{i,J,k}\rangle$ for each $(i,k)$ must be consecutive, with any 0's lying to the left or right end points of this string.

### 2.2 Containment

The condition of containment states that regions in $A$ must contain 12 distinct pitch classes. Let $B_p$ $(0 \leq p \leq 11)$ be the set of all locations $(i,j)$ of pitch class $p$ in matrix $A$. From this, we can express the condition of containment by the following:

$$\forall p \in [0,11], \forall k \in [1,K], \quad \sum_{(i,j)\in B_p} x_{i,j,k} = 1, \tag{3}$$

where for each $k$, $x_{i,j,k}$ is equal to 1 at one and only one location $(i,j)$ whose pitch class is $p$ in $A$. When this is the case, $C_k$ will contain one of each pitch class.

### 2.3 Covering all $(i,j)$ in $A$

A solution to our problem requires that every one element in $A$ is covered by at least one of the regions, $C_k$. We can express this condition by the following constraint:

$$\forall i \in [1,I], \forall j \in [1,J], \ \bigvee_{k=1}^{K} (x_{i,j,k} = 1). \tag{4}$$

### 2.4 Restrictions on the left-to-right order of candidate sets and their overlaps

As discussed in section 1, adjacent regions need not be contiguous in each row in $A$, however, there are restrictions on their left-to-right order and allowed

overlaps. The number of overlaps in each row between these regions must not exceed 1. We can express this restriction by the following constraint:

$$\forall i \in [1, I], \forall j \in [2, J], \forall k \in [1, K-1], \\ (x_{i,j,k} = 1) \Longrightarrow \bigwedge_{k'=k+1}^{K} (x_{i,j-1,k'} = 0). \tag{5}$$

When combined with the constraint of consecutiveness, constraint 5 means that if $x_{i,j,k}$ is equal to 1, the $i$th row of $C_{k'}$, whose $k'$ is greater than $k$, is either (1) located at the right-hand side of $(i, j)$ without overlapping the $i$th row of $C_k$ or (2) has only one overlap at the right-most element of the $i$th row of $C_k$.

### 2.5 Candidate sets as all different integer partitions

In order to determine that the integer composition "shape" of $C_k$ is a distinct integer partition, we introduce two variables, $y_{i,k,l}$ and $z_{k,l}$. Let $y_{i,k,l}$ be a binary variable that indicates whether or not the length of the $i$th row of $C_k$ is greater than or equal to $l$ ($1 \leq l \leq L, L = 12$), by introducing the following two constraints:

$$\forall i \in [1, I], \forall k \in [1, K], \ \sum_{j=1}^{J} x_{i,j,k} = \sum_{l=1}^{L} y_{i,k,l} \tag{6}$$

$$\forall i \in [1, I], \forall k \in [1, K], \forall l \in [2, L], \ (y_{i,k,l} = 1) \Longrightarrow (y_{i,k,l-1} = 1). \tag{7}$$

Equation 6 states that the sum of all elements in $\langle y_{i,k,1}, y_{i,k,2}, \ldots, y_{i,k,L} \rangle$ is equal to the length of the $i$th row of $C_k$ while Equation 7 states that its elements equal to 1 begin in the first position and are consecutive. For example, when the length of the $i$th row of $C_k$ is 3, $\langle y_{i,k,1}, y_{i,k,2}, \ldots, y_{i,k,L} \rangle$ is $\langle 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. The total number of rows in $C_k$ whose lengths are greater than or equal to $l$ is given by $\sum_{i=1}^{I} y_{i,k,l}$. Let $z_{k,l}$ ($0 \leq z_{k,l} \leq I$) be an integer variable that is equal to $\sum_{i=1}^{I} y_{i,k,l}$ ($1 \leq l \leq L$) with the following constraint:

$$\forall k \in [1, K], \forall l \in [1, L], \ z_{k,l} = \sum_{i=1}^{I} y_{i,k,l}. \tag{8}$$

The ordered set of twelve values $z_{k,l}$ ($1 \leq l \leq L$) will then identify the type of integer partition corresponding to $C_k$. For example, when $z_{k,l}$ is $\langle 4, 4, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0 \rangle$, $C_k$ is $\mathrm{IntPart}_{12}(5, 3, 2, 2)$. We denote this set $z_{k,l}$ corresponding to an integer partition $n$ by $P_n = \langle P_{n,1}, P_{n,2}, \ldots P_{n,L} \rangle$ ($1 \leq n \leq N, N = 34$), where integer partitions appear in reverse lexicographical order, meaning that those containing the fewest parts and largest part lengths appear first. For example, $P_1$ is $\mathrm{IntPart}_{12}(12, 0, 0, 0)$ and $P_{34}$ is $\mathrm{IntPart}_{12}(3, 3, 3, 3)$. From this, we determine the integer composition shape of $C_k$ to be the integer partition $n$ by the following constraint:

$$\forall k \in [1, K], \forall n \in [1, N], \ (w_k = n) \Longleftrightarrow \bigwedge_{l=1}^{L} (z_{k,l} = P_{n,l}), \tag{9}$$

where $w_k$ $(1 \leq w_k \leq N)$ is an integer variable that indicates to which $P_n$ $C_k$ corresponds. We can now express the condition that all integer partitions are distinct by the constraint, $\textsc{AllDifferent}(w_1, w_2, \ldots, w_K)$.

## 3 Solution

In order to confirm that our formulation of this problem is accurate, we implemented our constraints described in section 2 and supplied these to a CP solver (Sugar v2-1-0 [12,13]). We first tried to solve for the whole matrix directly, however, we were unable to obtain a solution after a day of calculation. We decided instead, to divide the matrix into two, equally-sized halves and try solving for each in such a way that their re-joining would form a complete solution to the original problem. We made this division of the original matrix at $[1, I] \times [1, J/2]$. Columns 1 to $(J/2)$ then correspond to the first smaller matrix we denote by $A_1$ and columns $(J/2) + 1$ to $J$ correspond to the second smaller matrix we denote by $A_2$. We allocated $34/2 = 17$ integer partitions to be found in each.

With little modification, our constraints can be adapted to the solving of these sub-problems. These changes include modifying $B_p$ (in equation 3) to contain only the locations of pitch classes in either $A_1$ or $A_2$, setting $K$ to be the new number of partitions in each (i.e., 17) and $J$ to be their new column lengths $96/2 = 48$. Solutions to $A_1$ and $A_2$ in which no integer partition is used more than once and contains only pitch classes from one or the other matrix (but not both), collectively form a solution to the original problem. Due to its smaller size, we were able to find solutions beginning with $A_1$ over the course of a day, in which 506 were found. Naturally, solving for $A_1$ makes finding a solution in $A_2$ more difficult as the number of available partitions is now fewer, and in fact, all 506 solutions to $A_1$ made $A_2$ unsatisfiable. We noticed, however, that certain partitions in these 506 solutions e.g., $\text{IntPart}_{12}(3, 3, 3, 3)$ and $\text{IntPart}_{12}(4, 3, 3, 2)$ occurred far less frequently than others. It would be reasonable then to conclude that solutions in $A_1$ which contain the greatest number of these less frequently occurring partitions will make solving for $A_2$ more likely, as the fewer available partitions in $A_2$ now consist of a proportionally greater number of frequently occurring partitions. Therefore, we solved again for $A_1$, this time by arbitrarily restricting the domain of $w_k$ to exclude the top 6 most frequently occurring partitions and include the top 5 least frequently occurring partitions.

If we denote the subset of integers from $[1, 34]$ corresponding to the partitions found in this solution to $A_1$, $S$, then the domain of $w_k$ for possible solutions to $A_2$ becomes $[1, 34] \setminus S$. We then tried solving for $A_2$, under the assumption that its proportionally greater number of more frequently occurring partitions would make finding a solution easier. While this means we exclude possible solutions e.g., ones in which a rarely occurring partition occurs in $A_2$ or where a partition contains pitch classes from both $A_1$ and $A_2$, we were able to generate a complete solution in this way. Solving for $A_1$ took approx. 4 minutes while solving for $A_2$ took approx. 28 minutes. Table 1 shows the complete solution found using this method of re-joining $A_1$ and $A_2$.

| et37 | 62 | 4581 | 90 | | 7 | t6e | 23510498 | -867 | 2t | e31 | -1094 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 8940 | 15 | 32e6t7 | | | 859410t23 | | e67 | 549 | 10 | -0 | 8te27365 |
| 2516 | 9t | 0 | 87e4365t | 219 | e | 03847 | | 1t2 | 9653 | 784 | |
| | 74830e | 9 | 12 | t56e07348 | 6 | 5291 | t | 03e | -e478 | t6592 | |
| $4^3$ | $62^3$ | $641^2$ | $82^2$ | 93 | $91^3$ | 543 | 831 | $3^4$ | $4^2 2^2$ | $53^2 1$ | 84 |

| 85637 | | -72e | t8 | 01945 | 32 | | 7et6890 | 514 | -4e2 | -2t36795481 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | -58 | 49013 | -3et276 | 450891et7 | 26 | 3 | | -30 | -0 |
| -4e09t | -t5126 | 430 | 7e | 8 | 6 | 95t1 | -124 | 0e3872 | 16t9578 | e |
| -21 | 4380e79 | -9t16 | -625 | | | 304e87 | 5 | 9t6 | | |
| $5^2 2$ | 75 | $43^2 2$ | $532^2$ | 651 | 921 | 642 | $731^2$ | $63^2$ | 732 | $10\ 1^2$ |

| 023t67e | | -e | 9 | | 8504 | 1 | 2e3t76 | 48 | 9501 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 9185 | 42673te10598 | -84 | 67 | | -7 | t3e29 | -908145 | 73 | 26et | |
| 4 | | 0396t5 | -5 | 21 | e3 | 40785 | | 619t20e | 837 | 4 |
| | | 127 | 83e0421t | -t5964738e0 | 2t916 | -6 | | 5 | 4 | 380e79t1625 |
| 741 | 12 | 6321 | $821^2$ | $10\ 2$ | 5421 | $5^2 1^2$ | $6^2$ | $72^2 1$ | $4^2 31$ | $11\ 1$ |

Table 1: A generated all-partition array corresponding to a complete solution to our problem, represented in the way used by music theorists [2]. Each column contains the elements in $A$ belonging to $C_k$, where a dash indicates those that overlap. Note, that partitions are denoted using a shorthand notation, e.g., $4^3$, where the base indicates the length of a part and the exponent denotes its number of occurrences. For clarity, the integers 10 and 11 have been replaced by the letters t and e, respectively.

## 4 Conclusion

In this paper, we have introduced a novel formulation of one part of the problem of generating an all-partition array, beginning from a given matrix, using constraint programming (CP). Solving for the whole of this matrix directly proved too difficult using our constraints. Therefore, we introduced a method of dividing the matrix into two halves, solving for each and then re-joining them to form a complete solution. Using this method, we were able to discover a solution. This solution is the first we are aware of to be automatically generated by a computer. Moreover, it is an all-together new all-partition array from those previously discovered by Babbitt and other composers. In future work, we hope to examine in more detail how to make finding solutions in larger matrices possible and without excluding potential solutions.

**Acknowledgments.** The work of Tsubasa Tanaka reported in this paper was supported by JSPS Postdoctoral Fellowships for Research Abroad. The work of Brian Bemman and David Meredith was carried out as part of the project Lrn2Cre8, which acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET grant number 610859.

## References


1. Babbitt, M.: Since Schoenberg. *Perspectives of New Music*, 12(1/2), 3–28 (1973).
2. Mead, A.: *An Introduction to the Music of Milton Babbitt.* Princeton University Press, Princeton, NJ, (1994).
3. Anders T., Anagnostopoulou, C., and Alcorn M.: Strasheela: Design and Usage of a Music Composition Environment Based on the Oz Programming Model. In Roy, P. editor, *Multiparadigm Programming in Mozart/OZ: Second International Conference, MOZ 2004, LNCS 3389*, Springer-Verlag, (2005).
4. Laurson, M. and Kuuskankare, M.: A constraint based approach to musical textures and instrumental writing. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop*, (2001).
5. Carpentier, G., Assayag, G. and Saint-James, E.: Solving the Musical Orchestration Problem using Multiobjective Constrained Optimization with a Genetic Local Search Approach. *Heuristics* 16(5), 681–714, Springer, (2010).
6. Chemillier, M. and Truchet, C.: Two musical CSPs. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop*, (2001).
7. Puget, J. F. and Régin J. C.: Solving the All Interval Problem `https://ianm.host.cs.st-andrews.ac.uk/CSPLib/prob/prob007/puget.pdf`
8. Bemman, B. and Meredith, D.: Generating Milton Babbitt's all-partition arrays. *Journal of New Music Research*, 45(2), (2016a). `http://www.tandfonline.com/doi/full/10.1080/09298215.2016.1172646`
9. Starr, D. and Morris, R.: A general theory of combinatoriality and the aggregate, part 1. *Perspectives of New Music*, 16(1), 3–35 (1977).
10. Starr, D. and Morris, R.: A general theory of combinatoriality and the aggregate, part 2. *Perspectives of New Music*, 16(2), 50–84 (1978).
11. Bazelow, A. R. and Brickle, F.: A combinatorial problem in music theory: Babbitt's partition problem (I). *Annals of the New York Academy of Sciences*, 319(1), 47–63 (1979).
12. `http://bach.istc.kobe-u.ac.jp/sugar/`
13. Naoyuki, T. and Mutsunori B.: Sugar: A CSP to SAT Translator Based on Order Encoding, in *Proceedings of the 2nd International CSP Solver Competition*, 65–69 (2008).