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.

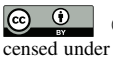
1. INTRODUCTION

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 6×96 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 6×12 excerpt from a 6×96 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 6×12 excerpt from a 6×96 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 6×12 excerpt from a 6×96 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 6×96 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

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 6×12 excerpt from a 6×96 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.

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.

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 kth 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:

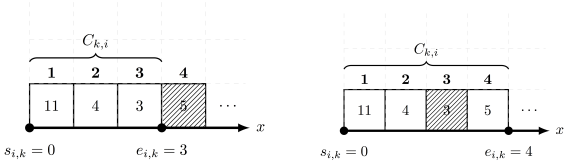
$$\forall k \in [1, K],\ \sum_{i=1}^{I} \sum_{j=1}^{J} x_{i,j,k} = 12, \tag{1}$$

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

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 ith-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.

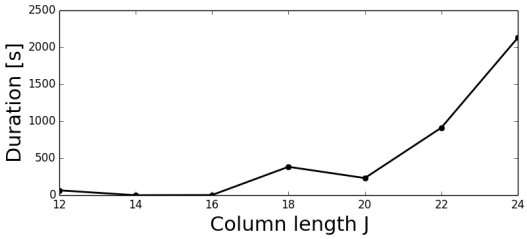


Figure 5: Duration of time spent on finding the first solution for each small matrix, whose column length is $J (12 \leq J \leq 24, J \in 2\mathbb{N})$. K is set to $(J + 2)/2$, resulting in 12 overlaps. Note, that no feasible solution exists when $J = 14$.

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.