Constraint Programming Approach to the Problem of Generating Milton Babbitt's All-Partition Arrays

Tsubasa Tanaka, Brian Bemman, David Meredith

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 4×96, 6×96, and 12×72 [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 4×96 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 4×12 excerpt from a 4×96 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 $L = \sum_{i=1}^{I} s_i$.

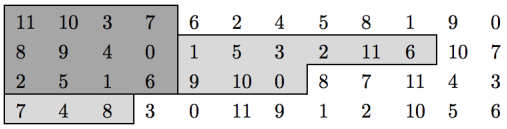

Fig. 1: A 4×12 excerpt from a 4×96 pitch-class matrix with two distinct integer partition regions represented precisely by the integer compositions, $\text{IntComp}_{12}(4, 4, 4, 0)$ (in dark gray) and $\text{IntComp}_{12}(0, 6, 3, 3)$ (in light gray), each containing every pitch class exactly once.

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 4×96 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

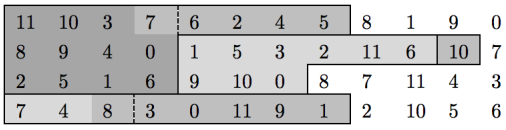

Fig. 2: A 4×12 excerpt from a 4×96 pitch-class matrix with a third integer composition, $\text{IntComp}_{12}(5, 1, 0, 6)$ (in medium gray), sharing one pitch class from each of the two previous regions.

(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 kth 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 ith 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 ith 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 ith row of C_k or (2) has only one overlap at the right-most element of the ith 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 ith 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 ith 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 ith 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).