# From analysis to surface

*Generating the surface of Milton Babbitt's Sheer Pluck from a parsimonious encoding of an analysis of its pitch-class structure*

Bemman, Brian; Meredith, David

*Publication date:*
2014

*Document Version*
Early version, also known as pre-print

# FROM ANALYSIS TO SURFACE: GENERATING THE SURFACE OF MILTON BABBITT'S *SHEER PLUCK* FROM A PARSIMONIOUS ENCODING OF AN ANALYSIS OF ITS PITCH CLASS STRUCTURE

**Brian M. Bemman**
Aalborg University
bb@create.aau.dk

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

## ABSTRACT

In recent years, a significant body of research has focused on developing algorithms for computing analyses of musical works automatically from encodings of these works' surfaces [3,4,7,10,11]. The quality of the output of such analysis algorithms is typically evaluated by comparing it with a "ground truth" analysis of the same music produced by a human expert (see, in particular, [5]).

In this paper, we explore the problem of generating an encoding of the musical surface of a work automatically from a systematic encoding of an analysis. The ability to do this depends on one having an effective (i.e., computable), correct and complete description of some aspect of the structure of the music. Generating the surface structure of a piece from an analysis in this manner serves as a proof of the analysis' correctness, effectiveness and completeness.

We present a reductive analysis of *Sheer Pluck* (1984), a twelve-tone composition for guitar by Milton Babbitt (1916–2011). This analysis focuses on the all-partition array structure on which the piece is based. Having presented this analysis, we formalize some constraints on the structure of the piece and explore some computational difficulties in automating the generation of the all-partition array structure.



## 1. INTRODUCTION

An *all-partition array* is a twelve-tone musical structure developed by Milton Babbitt that forms the basis of a number of compositions, particularly from his second period of works (1964–1980), although he continued to use this structure throughout his life. Thorough music-theoretical discussions and mathematical proofs of aspects of this structure in Babbitt's music can be found in the literature [2,6,8,9].

In essence, an all-partition array is a structure of tone rows that are organized into hexachordally combinatorial pairs and then partitioned into discrete, vertical aggregates. Each aggregate results from a distinct permutation of partitioned segments and can be represented precisely as an integer composition or, more abstractly, as an integer partition.[1] A six-part, all-partition array will have 58 such integer partitions.

In the first part of this paper, we provide basic definitions of concepts and terminology relating to the structure of all-partition arrays. The later  of the paper formalize row pairing constraints specific to one type of six-part, all-partition array and identify a particular computational difficulty in automatically generating this structure.

## 2. DEFINITIONS AND TERMINOLOGY

We define a *tone row*, $A = (a_0, a_1, a_2, \dots, a_{11})$, to be an ordered set of 12 distinct pitch classes in a system of 12-fold octave division. That is,

$$\bigcup_{a \in A} \{a\} = \{0,1,2 \dots 11\}.$$

Suppose that $A$ and $B$ are tone rows such that $A = (a_0, a_1, a_2, \dots, a_{11})$ and $B = (b_0, b_1, b_2, \dots, b_{11})$. We say that $A$ and $B$ are *hexachordally combinatorial*, denoted by $A\ h\ B$, if and only if

$$\{a_0, a_1, \dots, a_5\} = \{b_6, b_7, \dots, b_{11}\}.$$

Note that the two structures in this equality are *unordered* sets. That is, the pitch classes in the two hexachords do

[1] See also Young tableaux for alternate representations of these [2,12].

not have to appear in the same order in each row, $A$ and $B$.

If we define the pitch class *aggregate*, $U = \{0, 1, 2, \ldots, 11\}$, then, for any pitch class set, $x \subseteq U$, we define the *complement* of $x$ to be $\bar{x} = U \setminus x$.

We say that two tone rows, $A = (a_0, a_1, a_2, \ldots, a_{11})$ and $B = (b_0, b_1, b_2, \ldots, b_{11})$, are members of the same *row type* iff

$$\{a_0, a_1, a_2, a_3, a_4, a_5\}=\{b_0, b_1, b_2, b_3, b_4, b_5\}.$$

We denote the row type of a tone row $A = (a_0, a_1, a_2, \ldots, a_{11})$ by $(x, \bar{x})$ where $x = (a_0, a_1, a_2, \ldots, a_5)$.

We define an *integer partition*, denoted by IntPart($s_1$,$s_2$,...$s_k$), to be a representation of an integer $n = \sum_{i=1}^{k} s_i$, as an *unordered* sum of $k$ positive integers. For example, if $n$ = 12 and $k$ = 6, then one possible integer partition is IntPart(3,3,2,2,1,1) which is equal to IntPart(3,2,1,3,2,1) since integer partitions are unordered.

We define an *integer composition,* denoted by IntComp($s_1$,$s_2$,...$s_k$), to be a representation of an integer $n = \sum_{i=1}^{k} s_i$, as an *ordered* sum of $k$ *positive* integers. For example, if $n$ = 12 and $k$ = 6, then

$$\text{IntComp}(3,3,2,2,1,1) \neq \text{IntComp}(3,2,1,3,2,1).$$

Note that these two integer compositions are unequal because of the different ordering of the summands (or *parts*) in each.

We define a *weak integer composition,* WIntComp($s_1$,$s_2$,...$s_k$), to be a representation of an integer $n = \sum_{i=1}^{k} s_i$, as an *ordered* sum of $k$ *non-negative* integers. For example, if $n$ = 12 and $k$ = 6, then WIntComp(6,6,0,0,0,0) is a weak integer composition. Note that, because weak integer compositions are ordered sets, WIntComp(6,6,0,0,0,0) $\neq$ WIntComp(0,6,0,6,0,0). The difference between a weak integer composition and an integer composition is therefore simply that zeros are permitted in the former, but not the latter.

Within a particular context, we may choose to *bound* or *restrict* the number $k$ of parts or summands in an integer composition or partition. For example, within the context of the six-part all-partition array structure, we choose to bound the number of summands to six.

## 3. SIX-PART ALL-PARTITION ARRAYS

Typically, in a six-part all-partition array, Babbitt makes use of all 48 tone rows in a so-called *Babbitt square*.[2] As shown in Figure 1, in such an all-partition array, these 48 tone rows appear in a 6x8 grid. The rows and columns of this grid are typically referred to in the literature as *lynes* and *blocks*, respectively [6].

blocks →

lynes ↓

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $RI_6$ | $P_5$ | $I_9$ | $R_8$ | $I_3$ | $P_{11}$ | $RI_0$ | $R_2$ |
| $I_6$ | $R_5$ | $RI_3$ | $P_2$ | $RI_9$ | $R_{11}$ | $I_0$ | $P_8$ |
| $I_5$ | $P_1$ | $R_{10}$ | $RI_2$ | $R_4$ | $P_7$ | $I_{11}$ | $RI_8$ |
| $R_7$ | $RI_{11}$ | $I_2$ | $P_{10}$ | $I_8$ | $RI_5$ | $R_1$ | $P_4$ |
| $RI_7$ | $I_4$ | $R_9$ | $P_6$ | $R_3$ | $I_{10}$ | $R_{11}$ | $P_0$ |
| $R_6$ | $P_3$ | $RI_4$ | $I_1$ | $RI_{10}$ | $P_9$ | $R_0$ | $I_7$ |

**Figure 1**. 48 tone rows of a Babbitt square mapped to a 6x8 grid representing a 6-part all-partition array structure.

A *lyne* is a concatenation of tone rows, often of the same row type, while each block is a set of vertically aligned tone rows, one from each lyne. In the final surface structure of a composition, elision and repetition of pitch classes across block boundaries serves to obscure them and make the divisions between blocks more ambiguous than may be suggested by Figure 1. There are 48! ways in which the 48 standard transformations of a tone row can be mapped to this grid. However, in practice, this is severely constrained by the hexachordal combinatoriality relation, $h$ (defined above). The six lynes of the grid (each containing rows of a different row type) are grouped into three pairs. Within each pair of lynes, the rows in one lyne are $h$-related to the rows in the other lyne. Extending our terminology, we say that such lyne pairs are $h$-related. Figure 2 shows a representative block in a six-part all-partition array with three pairs of $h$-related rows.

$$\begin{array}{r|l|l}
\text{Lyne } (x,\bar{x}) & (\overbrace{11, 4, 3, 5, 9, 10}^{\{x\}}, 1, 8, 2, 0, 7, 6) & \ldots \\
\text{Lyne } (\bar{x},x) & (6, 7, 0, 2, 8, 1, 10, 9, 5, 3, 4, 11) & \\
\end{array}$$

$$\begin{array}{r|l|l}
\text{Lyne } (y,\bar{y}) & (\overbrace{5, 6, 11, 1, 7, 0}^{\{y\}}, 9, 8, 4, 2, 3, 10) & \ldots \\
\text{Lyne } (\bar{y},y) & (2, 9, 10, 8, 4, 3, 0, 5, 11,1, 6, 7) & \\
\end{array}$$

$$\begin{array}{r|l|l}
\text{Lyne } (z,\bar{z}) & (\overbrace{0, 5, 4, 6, 10, 11}^{\{z\}}, 2, 9, 3, 1, 8, 7) & \ldots \\
\text{Lyne } (\bar{z},z) & (1, 8, 9, 7, 3, 2, 11, 4, 10, 0, 5, 6) & \\
\end{array}$$

**Figure 2**. Typical single block within a six-part all-partition array. Note the three $h$-related row pairs.

[2] A *Babbitt square* is a collection of 48 tone rows equivalent to one another by some twelve-tone operation, P (transposition), I (inversion), R (retrograde), or RI (retrograde-inversion) [1].

It can be shown that there are $(8!)^6$ ways of organizing a 6x8 grid of tone rows, arranged as three pairs of $h$-related lynes. This still unwieldy number of possibilities is, however, further reduced by application of additional constraints that will now be described.

# 4. ROW PAIRING CONSTRAINTS IN *SHEER PLUCK*

## 4.1 Reductive Analysis

An alternative representation of the grid in Figure 1 is presented in Figure 3. This music-theoretical analysis reveals in more detail the row pairing constraints and relationships between rows. Each box contains a pair of $h$-related rows, each represented by the operation that generates that row from $P_0$ (e.g., ($RI_6$, $I_6$) in block I in lynes 1 and 2). At the top of each box is a header containing an operation that relates that pair of $h$-related rows (e.g., "R" for block I, lynes 1 and 2). The arrows labelled $T_3$ and $T_9$ indicate a cross-complement relationship between pairs of adjacent rows in one lyne to pairs of adjacent rows in another. Note also that blocks have $T_6$-related partners, represented by the arrows in the center of the diagram.[3]

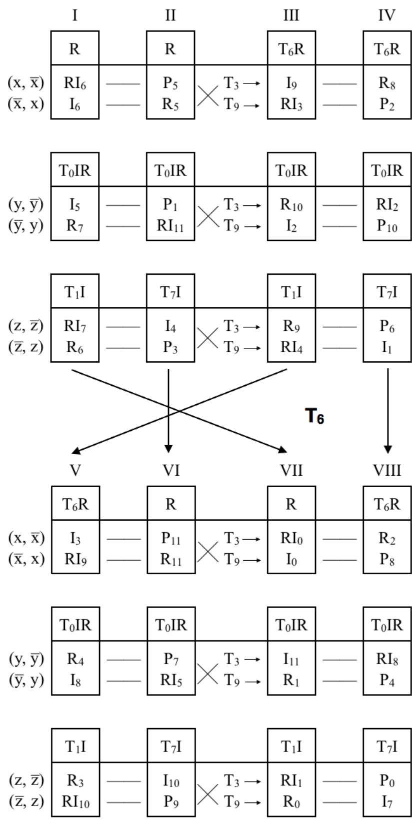


Figure 3. *Sheer Pluck* row pairing constraints and relationships.

[3] As noted in [6], this type of six-part, all-partition array was originally constructed by mathematician and composer, David Smalley. It is not clear the extent to which Babbitt contributed to its particular pairing of rows and sequence of integer compositions.

## 4.2 Formal Constraints

For a program to pair rows according to the constraints of Figure 3, these constraints must first be formally defined.

We begin by setting $(A, B, C, D) = (P, I, R, RI)$, for convenience. Let $(x,y)$ be any pair of row operations with unspecified transposition (level) in the same block in an $h$-related lyne pair (e.g., (RI,I), (I,R),(RI,R) in block I). The first condition that must be satisfed by all $(x,y)$ is that

$$(x, y) \in \{(x, y): x \neq y \wedge (x, y) \subset \{A, B, C, D\}\}. \quad (1)$$

We denote by $(x_i,y_i)$ any pair of $h$-related rows in the same block in lynes $i$+1 and $i$+2. We can then state the following three related conditions that are satisfied by such row pairs:

$$(x_0, y_0) \in \{(x, y): \{x, y\} \in \{\{A, C\}, \{B, D\}\}\}, \quad (2)$$

$$(x_1, y_1) \in \{(x, y): \{x, y\} \in \{\{A, D\}, \{B, C\}\}\}, \text{ and} \quad (3)$$

$$(x_2, y_2) \in \{(x, y): \{x, y\} \in \{\{A, B\}, \{C, D\}\}\}. \quad (4)$$

Finally, if $p$ and $q$ are adjacent blocks and $p = ((x_0,y_0), (x_1,y_1), (x_2,y_2))$, then $q = (((x'_0,y'_0), (x'_1,y'_1), (x'_2,y'_2))$ where

$$\{x'_i,y'_i\}=\{A,B,C,D\} \setminus \{x_i,y_i\}. \quad (5)$$

Constraints (1), (2), and (5) are visualized in Figure 4.

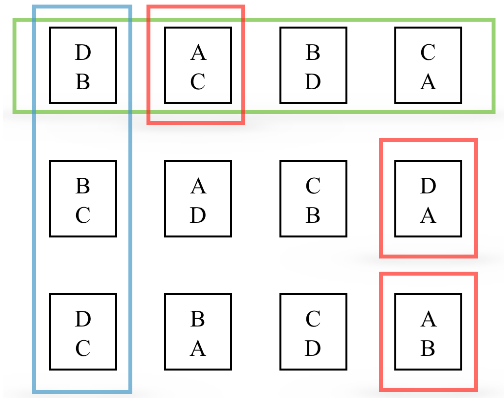


**Figure 4**. First four blocks of *Sheer Pluck*. Red (1), Green (2) and Blue (5).

Pairing rows according to the constraints listed above now reduces the number of possibilities for tone row organization to a much more manageable 96. This, however, represents only one requirement of the all-partition array. The second, parsing this structure into vertical aggregates, is a more difficult task.

# 5. PARSING INTO VERTICAL AGGREGATES

Discrete, vertical aggregates are distinguished according to the partitioning of members from each lyne into segments of length 12 or fewer. These segments can be represented abstractly (without regard for order) as integer partitions, but when realized, are more precisely represented (i.e., with regard for order) as integer compositions. Figure 5 shows one example.

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

**Figure 5**. Integer partitions $52^21^3$, 4 8, and $71^5$ realized as integer compositions in an incomplete single block.

The number of integer partitions present in an all-partition array is equal to the number of partitions of 12 given $k$, the number of summands, as defined above. A $1^{12}$ partition (where $k = 12$) for example, is not available to a six-part array because it contains only six lynes (where $k = 6$). Of the total possible 77 integer partitions of 12, a six-part array will therefore contain only 58 [8]. Of these 58 required integer partitions, there are 6,188 possible integer composition subsets to choose from (i.e., the permutations of each integer partition).

The 58 integer partitions we need to parse a 6x8 grid require 696 pitch classes. However, our grid of 48 tone rows contains only 576 pitch classes. Therefore, the inclusion of an additional 120 pitch classes (examples of which are shown in bold in Figure 5) is necessary in order to successfully parse it. Determining where to place these extra pitch classes is not trivial, because of the very large number of possible ways of placing $n$ distinct objects into $k$ distinct locations. For example, when $n = 120$ and $k = 48$, then the number of possible solutions is $48^{120} \approx 5.61 \cdot 10^{201}$. Clearly, it would thus be impossible to exhaustively search through all these possibilities. One of our goals is to find an computationally tractable solution to this problem.

## 6. CONCLUSION

As shown above, having a program complete the first step of constructing an all-partition array (organizing $h$-related rows) is relatively straightforward. The computational difficulties arise, however, when attempting to parse it. Without providing a program with the correct locations of the required extra 120 pitch classes, an exponential growth in computing time will occur with an exhaustive search of their possible placements. Even if the location of these pitch classes is provided, a similarly exhaustive search for possible integer compositions is not possible, as it too will result in an exponential growth in computing time. We are currently exploring the possibility of using a greedy approach to solve this problem.

## 7. REFERENCES


[1] Babbitt, M. (1961). Some aspects of twelve-tone composition. International Music Association. p. 61.

[2] Bazelov, A. R. & Brickle, F. (1976). A partition problem posed by Milton Babbitt (Part 1). Perspectives of New Music, 14(2), 280–293.

[3] Cambouropoulos, E. (2008). Voice and stream: Perceptual and computational modeling of voice separation. Music Perception, 26(1), 75–94.

[4] Chew, E. (2014). Mathematical and Computational Modeling of Tonality: Theory and Applications. New York: Springer.

[5] Collins, T. (2013). MIREX 2013 Competition on Discovery of Repeated Themes and Sections. Curitiba, Brazil. http://www.music-ir.org/mirex/wiki/2013:Discovery_of_Repeated_Themes_ %26_Sections

[6] Mead, A. W. (1994). An Introduction to the Music of Milton Babbitt. Princeton, NJ: Princeton University Press.

[7] Meredith, D., Lemström, K. & Wiggins, G. A. (2002). Algorithms for discovering repeated patterns in multidimensional representations of polyphonic music. Journal of New Music Research, 31(4), 321–345.

[8] Morris, R. D. & Alegant, B. (1988). The even partitions in twelve-tone music. Music Theory Spectrum, 10, 74–101.

[9] Starr, D. & Morris, R. (1978). A general theory of combinatoriality and the aggregate (part 2). Perspectives of New Music, 16(2), 50–84.

[10] Temperley, D. (2001). The Cognition of Basic Musical Structures. Cambridge, MA: The MIT Press.

[11] Temperley, D. (2007). Music and Probability. Cambridge, MA: The MIT Press.

[12] Yong, A. (2007). What is … a Young Tableau?. Notices of the AMS, 54(2), 240–241.