Proceedings of the 12th Brazilian Symposium on Computer Music
Anais do 12º Simpósio Brasileiro de Computação Musical

NUCOM – Comissão Especial de Computação Musical da SBC
Sociedade Brasileira de Computação
http://compmus.ime.usp.br/sbcm/

7-9 September, 2009
Recife – PE – Brazil
Promoção / Promotion:

NUCOM/CECM Comissão Especial de Computação Musical da SBC

Organização / Organization:

USP UFRGS UFAC UFPE

Suporte / Sponsorship:

CNPq CAPES FACEPE RBV

Apoio / Institutional support:

PORTODIGITAL DOTTI EMF Centre Pompidou D’Accord Music Software GOL

Proceedings of the 12th Brazilian Symposium on Computer Music,
September 7-9, 2009, Recife, Brazil / eds. Marcelo Pimenta, Damián Keller,
Regis Faria, Marcelo Queiroz, Geber Ramalho, Giordano Cabral ; org. tec. Mar-
celo Pimenta ; org. mus. Damián Keller ; org. local Geber Ramalho e Giordano
Cabral ; org. geral Regis Faria e Marcelo Queiroz. – São Paulo : USP : SBC,
2009.
on-line + CD-ROM
ISSN 2175-6759
http://compmus.ime.usp.br/sbcm/

1. Computação musical (Congressos) 2. Computação aplicada
(Congressos) 3. Música computacional (Congressos) I. Sociedade Brasileira de
Computação II. Universidade de São Paulo III. Universidade Federal do Rio
Grande do Sul IV. Universidade Federal do Acre V. Universidade Federal de
Pernambuco VI. t.

CDD 004 786.76
Pitch-class composition in the *pd* environment

Pablo Cetta¹, Oscar Pablo Di Liscia²

¹ Universidad Católica Argentina, Facultad de Artes y Ciencias Musicales, Buenos Aires, Argentina.
² Universidad Nacional de Quilmes, Carrera de Composición con Medios Electroacústicos, Bernal, Argentina.

Pablo_cetta@uca.edu.ar, odiliscia@unq.edu.ar

Abstract. This paper presents research on composition-assisted software using the Pitch-Class Sets technique [Babbitt, 1961] [Forte, 1974]. In order to use the mentioned technique on music composition and analysis, a Library of External Objects for the program Pure Data [Puckette, 2009] was developed. The Pitch-Class Set technique uses both the combinatorial and set theory to organize the twelve Pitch-Classes of the tempered system in Sets in order to exploit their structural properties on atonal music composition and analysis. A latter projection of this system explores the possibilities of disposition of the Pitch-Class Sets in the musical space producing Combinatorial Matrices [Morris, 1984, 1987].

1. Presentation

The Pitch-Class Set technique uses both the combinatorial and set theory to organize the twelve Pitch-Classes of the tempered system in Sets (*Pitch-Class Set* will be from here on abbreviated as PCS) in order to exploit their structural properties on atonal music composition and analysis. Although it is clearly granted that this system was inspired on the European pre and post serial atonal music, it was initially developed by American composers and theorists [Babbitt, 1961] [Forte, 1974].

1.1 Pitch Class Sets.

Generally speaking, the PCSs technique covers three aspects: Taxonomy, Properties and Relations. The first aspect deals with the concept of the PCS as a subset of the Universal Superset formed by the twelve pitch-classes (also called “aggregate”) and the concepts of equivalence by inversion-transposition that generate the 224 different Set-Classes (*Set-Class* will be from here abbreviated as SC). The second defines, encodes, analyzes and classifies the structural features of each SC (as is, for example, its Interval Class Vector). The third deals with the possible relations between PCSs and SCs and their significance in the musical context (see, [Forte, 1974], [Morris, 1980], [Isaacson, 1990], [Rahn, 1980], among others).

1.2 Combinatorial Matrices.

A latter projection of this system explores the possibilities of disposition of PCSs in the musical space producing *Combinatorial Matrices* (*Combinatorial Matrix* will be from here on abbreviated as CM) and creating abstract compositional designs [Morris, 1984, 1987]. Such approach is based, of course, on the assumption that it is possible to extend some of the properties and relations of the PCSs to successions and combinations of them, a fact that is—in the opinion of the authors—proven by both the music and the theory of atonal music. In the theoretical works of Morris [Morris, 1984, 1987], the CMs are two-dimensional combinations of PCSs where the constancy of one or two SCs
is regarded as a cohesive factor, as it may provide some sonic uniformity. The extension of this
criterion to more than two PCSs is also possible, being the easiest way of accomplishing it the
combination (by juxtaposition or superposition) of the “simple” CMs. This possibility will be
shown and explained in the final section of this paper.

1.3 And what about music?
An outstanding number of significant relations of the PCSs theory with the organizations of other
sound or music properties (such as register, duration, intensity and timbre, among others) have been
already developed in serial music and its derivations (See, among other, [Stockhausen, 1959],
[Babbitt, 1961, 1962], [Mead, 1987], [Roming, 2000]). These relations provide an important basis
for the production of meaningful music, as they may allow the production of a truly multi-
dimensional musical space.

2-Computer applications of the theory
The complexity of the atonal theory makes its practical application almost impossible without the
aid of computer applications. As a matter of fact, the PCSs technique and its extensions seem to be
developed in synergy with the software for musical composition and analysis. This research and
development approach was followed in the project “Desarrollo de aplicaciones informáticas para
la organización de la altura temperada en la composición y análisis musical” (Area Trans
Departamental de Artes Multimediales, Instituto Universitario Nacional del Arte, 2007-2008). The
results of the projects are several computer applications, publications, documentation and data that
may be obtained at: http://www.iuna.edu.ar/departamentos/multimedia/investiga/sitio_analisis/

One of the computer applications developed in this project is the pcslib library to be used in
the pd environment (Pure Data, [Puckette, 2009]). pcslib is a set of “external objects” that allows
the work with PCSs and CMs in the pd environment. Generally speaking, the developed objects are
software units that work with high level data types. The pcslib library is fully documented and some
eamples of the use of each object are provided as well. pcslib is an open source, and of free use,
copy, and distribution. Among other factors, the pd environment was selected because it allows
structured programming and because of its capability of handling Audio, MIDI and Graphic data in
real and non-real time. The first feature ensures the possibility of working with different levels of
data abstractions, which is especially useful in the applications of the PCSs and CMs theory to
composition.

2.1 PCS and CM numerical representation in pcslib.
Because of both computing requirements and theoretical conventions, pcslib uses a numerical
representation of the attributes of the PCSs and CMs which will be briefly described in this section.
Since the code of pcslib is available, a user who is experienced in c programming language can
find further details easily, if necessary.

A PCS is a set of Pitch Classes(PCs), each one of these being represented by one
integer(from 0 to 11). Usually, a PCS is considered as non-ordered, meaning that 0,1,2,3 is
equivalent to 0,2,1,3 and also to 0,1,3,2 and so on. However, in order to work properly with Chains
and CMs it is necessary to consider the PCSs as partially ordered, in the sense that a PCS may have
different groupings of its PCs referred to as positions. Such partially ordered PCS is referred to as a
chain. A position of a chain may have more than one PC as well.

For example, the same PCS {0,2,4,5} may be distributed forming different chains with
different number of positions(separated by spaces in what follows):
A CM (Combinatorial Matrix) is a bi-dimensional array of PCs. X being a CM, the content of each of its positions may be accessed at \( X_{row,column} \). The PCS content of each column and row of a CM must belong to the same SC or either to two different SCs [See Morris, 1984, 1987]. The simplest of the CMs is the one called roman square. An example of this CM is shown below:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Here we have a CM with 4 rows and 4 columns (4*4=16 positions) and only one PC in each position (which is, of course, not always the case).

A CM is represented by pcslib having as many vectors of 9 positions each as positions the CM have. Any position of each vector which is empty (i.e., with no PC in it) will be filled with a -4. For example, the former roman square CM shown above will be represented internally by pcslib as:

\[
\begin{bmatrix}
0 & -4 & -4 & 4 & -4 & 4 & 4 & 4 & 4 \\
-4 & -4 & -4 & 4 & 4 & 4 & 4 & 4 & 4 \\
-4 & 4 & -4 & 4 & 4 & 4 & 4 & 4 & 4 \\
-4 & 4 & 4 & -4 & 4 & 4 & 4 & 4 & 4 \\
-4 & 4 & 4 & 4 & -4 & 4 & 4 & 4 & 4 \\
-4 & 4 & 4 & 4 & 4 & -4 & 4 & 4 & 4 \\
-4 & 4 & 4 & 4 & 4 & 4 & -4 & 4 & 4 \\
-4 & 4 & 4 & 4 & 4 & 4 & 4 & -4 & 4 \\
-4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & -4 \\
\end{bmatrix}
\]

(PCs are marked boldface, and each [...] enclose the data representing a position of the CM)

A CM may have one or more positions with more than one PC in it, and/or one or more empty positions as well. For example, the 6X6 CM which follows combines positions of 3 and 2 PCs each. The CM shown below have several empty positions as well (positions [1][0], or [0][2], for example). An empty position is represented by pcslib as one -3 followed by eight -4 values. Note also that, in order to use only one character for each integer, an “A” replacing the integer 10 and a “B” replacing the integer 11 are used, but this is only for “printing neatness”, not for the internal representation of the data.

\[
\begin{bmatrix}
0 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
-4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 & -4 \\
\end{bmatrix}
\]
2.2 Special data types used by pcslib under pd:
In order to perform an efficient data transfer between its different objects, in addition to the pd usual
data types(like floats, lists, symbols, etc.), pcslib uses two special data types.

1-Pointers to structures of the PCS type.
2-Pointers to structures of the CM type.

Details of each type of structure can be found in the pcslib code. Of course, the data of each
one of these structures cannot be accessed directly, but there are objects especially dedicated to
create, read and modify them.

2.3 Summary of pcslib externals:
Due to the extension limitations of this paper, a detailed description of each of the 17 externals
objects of pcslib will not be done. Therefore, this section will only present a summary of them. Two
brief examples of use are provided in the next section.

Pitch-class sets(PCS) objects:
   pcs_write: The pcs_write object generates a PCS using data delivered by the user. The PCS
   is scanned at the PCS table and transposed and/or inverted, if required.
   pcs_read: The pcs_read object reads and classifies the data in a PCS structure.
   pcs_pf: The pcs_pf object finds the prime form of a PCS and stores all its associated data in
   a PCS structure.
   pcs_ttos: The pcs_ttos object performs the Transposition, Inversion followed by
   transposition or Multiplication operations on a PCS.
   pcs_subs: The pcs_subs object gets the all the subsets of cardinal n out of a given PCS.
   pcs_sim: The pcs_sim object evaluates the similarity degree of two SCs according to either
   Forte, Morris or Isaacson’s criteria(See [Forte, 1974], [Morris, 1980], [Isaacson, 1990]). In addition
   it also checks if the two PCSs belong to the same Kh complex(See [Forte, 1978]).
   pcs_invar: The pcs_invar object finds the shared PCs between two or more PCSs.
   pcs_parts: The pcs_parts object gets all the binary partitions out of a given PCS.

Combinatorial Matrix(CM) objects
   cm_roman: The cm_roman object creates a roman square type of CM(see [Morris, 1984,
   1987]).
   cm_t1a: The cm_t1a object creates a t1a type of CM(see [Morris, 1984, 1987]).
   cm_t1b: The cm_t1b object creates a t1b type of CM(see [Morris, 1984, 1987]).
   cm_t2: The cm_t2 object creates a t2 type of CM(see [Morris, 1984, 1987]).
   cm_opcy: The cm_opcy object creates a CM using cycles of operators(see [Morris, 1984,
   1987]).
The `cm_2txt` object has no output, only post to the `pd` prompt a "clean" i.e., not raw, version of a CM.

The `cm_read` object gets the “raw” data in a CM. -3 and -4 are spaces and empty positions, respectively.

The `cm_2pcs` object gets a PCS from a CM.

The `cm_trans` object performs several transformations on a given CM. The transformations that may be performed by this object are: transposing, inverting, rotating by the diagonal, rotating by 90 degrees, exchange of rows or columns content and swapping elements to decrease density.

### 2.4 Two examples

In order to show some of the `pcslib` potential, two examples are presented in this section. Although the output of the `pd` patches shown is numerical and highly abstract, the readers must keep in mind that the goal of `pcslib` is to provide a toolkit for generating a control-signal flow which may be useful to drive a digital musical system. The design of such musical systems, however, implies large patches using the MIDI and Audio capabilities of `pd` and is not the subject of this paper.

First, a simple `pd` patch using the `pcs_pf` and `pcs_read` externals is shown on Figure 1. This simple patch intends to show just one way in which the PCS data type may be created, transmitted and read in order to access its data members, as well as introduces the reader in some of the basic concepts of the PCSs theory. In the patch, a list containing Pitch-Classes is passed to the `pcs_pf` object, which creates a PCS whose pointer is passed to the `pcs_read` object. The latter object outputs the following data by its seven outlets (ordered from left to right):

**outlet1:** The original PCs of the PCS as delivered (`floats` list).

**outlet2:** The cardinal number (`float`), i.e., the number of different PCs that the PCS have.

**outlet3:** The ordinal number (`float`) of the PCS. A number indicating the SC to which the PCS belongs. The Atonal Theory uses the cardinal and the ordinal numbers separated by a hyphen to define the name of the PCS and its SC (See [Forte, 1974]). In the example shown, the PCS name is “5-17”.

**outlet4:** The status (T/I) (`symbol`). The transposition and/or inversion status of the PCS.

**outlet5:** The prime form of the PCS (`floats` list). The _prime form_ of a PCS is a version of it determined by convention (See [Forte, 1974]).

**outlet6:** The interval-class vector (`floats` list). The six floats delivered represent the interval contents (interval-class content) of the PCS. The data is to be interpreted as a six-position numerical vector. Thus, the first number is the amount of members of the interval-class 1 (minor second, or SC 2-1), the second one is the amount of members of the interval-class 2 (major second, or SC 2-2), and so on.

**outlet7:** The literal complement of the PCS, i.e., the PCs that are not included in it (`floats` list).

It is important to note that, once a PCS structure is created, all its data may be useful to perform more complex operations in a signal flow which may drive a musical system. Just as an example, a patch could be done in which a minimal number of members of any interval-class of a PCS is required to take an action. Another possibility could be to compare two or more PCSs (using the `pcs_sim` object) in order to check whether a certain kind of relation stands.
The second patch presented uses the combination of PCSs in order to generate a CM, and performs several transformations of it. Here, a list containing Pitch Classes is passed to the `pcs_pf` object, which creates a PCS whose pointer is passed to the `cm_opcy` object. The latter object creates a CM and passes its pointer to the `cm_trans` object which will perform the transformations that are requested by the messages that are allowed to process. The `cm_2txt` shows the resulting CM and its successive transformations on the `pd` prompt. The patch is shown in Figure 2.
In this case, a series of swapping operations was selected to get different variations of the original CM. The CMs resulting from the successive swapping operations are shown in Figure 3. It can be easily seen that the swapping operation is produced by the object \textit{cm\_trans} without changing the PCS content of the CM, thus producing a decrease in density (the amount of PCs in each position of the CM) while preserving the sonic uniformity.

The latter structures can be presented in different ways in order to generate a meaningful music sequence. Just to show one of them, the same time-interval is assigned to each PC in the CM, each one of its rows is assigned to a pitch register, and the beginning of each CM row is marked with an accent. The sequence shows an increasing density in simultaneity of notes and accents. Also, it can be seen that the time pattern of the accents is homogeneous at the beginning and the end of the sequence whilst it is non-homogeneous at its middle. The musical sequence is scored in two staves only to keep it easier for reading and is shown in Figure 4.
3-Conclusions

Though pcslib is at present still under development, an important number of external objects that are capable to perform operations of significant complexity have already been developed. The CM generation algorithms are effective and capable of generating a great number of them, but many of these are trivial and/or tautological. This does not mean that they are not useful, as it was shown by the examples that they constitute a strong base for creating pitch-organised music. It was also shown that, in order of having music and not merely PCSs and/or CMs, still higher level (i.e., more “close to music”) programming is needed. Some of the directions to be further explored in the latter approach may include the development and implementation of criteria for the selection and combination of CMs.
4-References


