Pitch-class Set design in SuperCollider

Lucas SAMARUGA
Universidad Nacional de Quilmes
Roque Saenz Peña 180
Quilmes, Argentina, 1876
lsamaruga@becarios.unq.edu.ar

Oscar Pablo DI LISCIA
Universidad Nacional de Quilmes
Roque Saenz Peña 180
Quilmes, Argentina, 1876
odiliscia@unq.edu.ar

Abstract

The *Pitch-class set* theory [6] and its extensions [8] constitute an important basis for mastering multi- layered atonal composition. The *SuperCollider* [10] environment offers significant possibilities of applying this technique in the creation of abstract *Pitch-class* designs that may be used as a part of more complex algorithmic composition developments. This paper presents *pcslib-sc* a *quark* (library) for *Pitch-class set* design in *SuperCollider* and a use case in order to demonstrate its musical relevance.

Keywords

Pitch-class Set Composition, SuperCollider, Musical Composition.

1 Introduction

The *Pitch-class Set* theory 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 abbreviated as PCS and *Pitch-class* as PC) in order to exploit their structural properties on atonal music composition and analysis. Although it is evident that this system was inspired on the European pre and post serial atonal music[1], it was initially developed by American composers and theorists like Milton Babbitt [1], and Allen Forte [6].

Generally speaking, the PCS theory covers three aspects. 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. The second defines, encodes, analyses and classifies the structural features of each set class (such as, for example, their Interval Class Vector). The third deals with the possible relations between PCSs and set classes and their significance in the musical context[2].

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[3].

The complexity of atonal theory makes its practical application almost impossible without the aid of computer applications. Therefore, one of the computer applications developed by the team of this project was the *pcslib* library (by Pablo Di Liscia and Pablo Cetta [2]) to be used in the *PD* environment (*Pure Data*, by Miller Puckette et al). *pcslib* is a set of “external objects” that allow the work with PCSs and CMs in the *PD* environment [4].

pcslib-sc for *SuperCollider*[4] is based on *pcslib* for *PD* with two small differences: 1) the adaption of the original library interface to a more general-purpose object-oriented language and 2) the generalisation of some functionality in relation to set theory.

Although *SuperCollider* is generally more oriented to real-time sound synthesis and algorithmic composition, its language is very useful in manipulating and analysing musical data mainly because of its dedicated library. Therefore this *quark* is intended to work with the structure generation approach of the PCS theory rather than the pattern generation, task that can be accomplished using the standard library.

This paper outlines the usage of three main resources of PCS theory, its related data structures and their combined use: *Pitch-class sets*, *Pitch-class chains* and *Pitch-class matrices*. The following discussion assumes that the reader is

[1] Mainly, the music of Arnold Schönberg, Anton Webern and Alban Berg, the three leading composers of the so-called Viennese School.

[2] For an extensive review on this specific subject, see Di Liscia [5].

[3] See Morris [7], [8].

[4] *pcslib-sc* was written by Lucas Samaruga under the supervision of Pablo Di Liscia and can be obtained at: https://github.com/smrg-lm/pcslib-sc.

aware of both the fundamentals of the PCS theory and of *SuperCollider* programming.

2 Data Structures

There are three main classes that are intended to work in combination for the elaboration of pitch structures. The *PCS* class, which defines a particular PCS (together with its properties and operations), the *PCSChain* class, which defines a *chain* of *PCSs* and its elaboration methods and the *PCSMatrix* class, which defines a *combinatorial matrix* of *PCS* (together with its particular properties, generation methods and operations).

The *PCS* class inherits from the library class *OrderedIdentitySet* not only because they share common set operations, but also because of the importance of the *Pitch-classes* order for some data calculation. *PCSChain* and *PCSMatrix* classes are more likely utility classes to work with their respective PCS theory counterpart. The former is a list and the latter represents a matrix of *PCS*. In their combined usage, chains are built from *PCS* and matrices can be built from *PCS* or chains. There is also a *SCTable* class, which holds the *Set-class* (SC)[5] table used for information retrieval from the *PCS* class.

These classes do not try to cover exhaustively all structural processes, but rather the common qualities used for compositional design. Therefore, the *PCS* class is the most complex in terms of information retrieval, operations and transformations, as they are the basic resources for further developments.

2.1 Basic Properties and Information

As stated before, the *PCS* class is the core class of the quark, it holds all the operations and properties related to the PCS theory.

A *PCS* can be built from its symbolic table name, consisting of its cardinal and ordinal numbers separated by a hyphen, like '4-16' or '5-12', but without the 'Z' pair identifier which can be queried with the *z* method, e.g. '5-12' is written '5-Z12' in Forte's nomenclature [6]; or can be created from an array of numbers, like *PCS[0, 1, 3, 5, 6]* with the array syntax shortcut. Internally, the *PCs* are stored as an ordered set in modulo 12, so the conversion of an array of MIDI notes such as *[77, 72, 54, 49, 51, 60, 51, 48]* will result in *PCS[5, 0, 6, 1, 3]*.

Once the note numbers are stored in a *PCS* its *prime form*, *normal order*, Forte's *name*, *Z pair*, *interval-class vector*, *invariance vector* and *cyclic adjacent interval array*, can be obtained from the *SCTable* through the *PCS* instance methods. Also the *twelve-tone operators*, *relations*, *similarity* and *status* between different *PCSs* and its *prime form* are supported as basic operations aside of the inherited set operations.

2.2 Combined Use and Design Methods

PCSChain and *PCSMatrix* classes are related to the pitch design in one and two dimensions respectively (see below). They holds the structured data as higher abstractions and provides different creation, information and manipulation methods.

PCSChain is used to build chains with the procedures explained in Section 3.2. The built lists can be used as streams by the pattern library.

PCSMatrix can be built with different methods: from arrays (a free matrix), from chains (special cases were the chain is specially set, as explained in Section 3.1), from SC to build different Morris's CM types, and from twelve-tone operators [8]. Basic operations like swapping, transposition, multiplications, inversion, rotation, and information about the properties of a CM, such as *sparseness* and *fragmentation* factors and *histogram* of PC density are also provided [8]. The data of the rows and columns of the matrix can be converted back to *PCS* or used as streams as well.

3 Use case: constructing Combinatorial Matrices from chains

In this section a use case, out of the many possible using *pcslib-sc*, will be presented with the objective of showing its musical relevance. The particular subject of PCS composition addressed here will be CMs. In the next section, the basic underliying theory of CMs is briefly explained.

3.1 Introduction to the theory of Combinatorial Matrices and chains

CMs are two-dimensional arrays that hold in their vertical and horizontal dimensions PCSs of one or more SCs. The classes of those PCSs are referred to as the *norm* of a CM and are meant to produce sonic coherence with respect to some particular pitch organization. As shown in [7] and [8], there are several methods to deal with the construction of CMs, and several CMs types. The method addressed here is the construction of *chains* of PCSs.

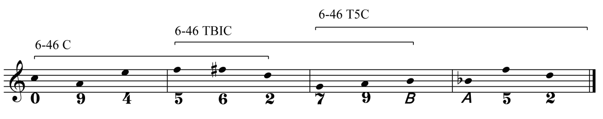

Figure 1

[5] In this paper SC stands for *Set-class*, and it should not be confused with *SuperCollider*, which is written always with no acronym.

A chain is a succession of PCSs that, being considered in adjacent pairs, form a PCS of a particular SC referred to as norm. An example of such a chain is presented in Figure 1. The ordered succession of the (unordered) PCSs: < {094} {562} {79B} {A52} >[6] constitutes a chain having the class 6-46 as its norm. The horizontal brackets show how the norm of the chain overlaps between the adjacent pairs of PCSs.

A chain with a unique norm may be taken as a basis for constructing a CM with the same norm. A chain constructed with the set class 5-15 together with its corresponding CM is shown below:

PCS chain: < {01} {268} {07} {15B} {67} {028} {16} {57B} >

Resulting CM (Table 1):

01	268		
	07	15B	
		67	028
57B			16

Table 1

As can be seen, the union of the PCS of each one of the columns and each one of the rows of the CM form a PCS of the class 5-15 (the norm of the CM). The distribution of the PCS in the resulting CM may be further improved through swapping operations[7]. One possible result would be (Table 2):

1	02	6	8
B	7	15	0
0	8	7	26
57	6	B	1

Table 2

3.2 Constructing chains

Essencially, it can be said that the method[8] for chain construction consists in connecting different transposed and/or inverted, partially-ordered versions of a PCS. Such partially ordered versions are the binary partitions of a PCS which will be termed *partitions*. For example, the Table 3 below shows the ten different partitions of a PCS of the class 5-15, and has all the information needed for constructing a chain with this set-class as norm:

If, for instance, the partition F (01|268) is selected for starting the chain, the different 2/3 transposed and/or inverted partitions having a PCS of cardinality 3 that match the PCS in the 'right part' of the starting partition are candidates for continuing the chain. When one of these candidates is selected and added to the chain, there will be new candidates to continue the chain according to the new PCS added, and the procedure is continued as explained until it is decided that the chain must be finished or when a partition that closes the chain is found[9].

5-15 {0,1,2,6,8}			
Partitions 1/4			
A	0|1268	1-1	4-16
B	1|0268	1-1	4-25
C	2|0168	1-1	4-16
D	6|0128	1-1	4-5
E	8|0126	1-1	4-5
Partitions 2/3			
F	01|268	2-1	3-8
G	02|168	2-2	3-9
H	06|128	2-6	3-5
I	08|126	2-4	3-4
J	12|068	2-1	3-8
K	16|028	2-5	3-8
L	18|026	2-5	3-8
M	26|018	2-4	3-4
N	28|016	2-6	3-5
O	68|012	2-2	3-1

Table 3

3.3 Choosing partition candidates

The explained method suggest that more than one candidate for continuing a chain may be found, depending on both the chain itself and the properties of the SC of its norm. If the norm does not change along the chain, then at least one candidate partition to continue it will exist[10]. When more than one candidate exists, one or several selection criteria must be applied. A criterion for measuring the 'qualification' of a list of candidates is to score them according their contribution of new PCs in the chain or, if the aggregate set is complete, according the distance of the PCs added

[6] The convention of representing PC 10 with an *A* and PC 11 with a *B* will be followed from here for practical reasons.

[7] Such swapping operations are documented in [7] and [8] and will not be explained here.

[8] The method is fully explained in [7] and [8]. See also [3].

[9] The possibility of finding a *partition* that may close the *chain* is explained in [7].

[10] This partition may not be musically interesting, as it is the same *partition* in reverse order,.

to their previous presentation[11]. Such criterion is formalized as:

$$score(cand_i)=\frac{\sum_{n=0}^{C_{i-1}} dist(pc_n, cand_i)}{(S)C_i}$$

EQ. 1

Where *S* is the chain size (number of positions); C_i is the cardinality of the *ith* PCS to be added; $dist(pc_n, cand_i) = S - (pos(pc_n, cand_i) + 2)$ and $pos(pc_n, cand_i)$ is the last position in which the pc_n of $cand_i$ was found (i=0 to S-1, and n=0 to C_i). If the pc_n is not found in the chain, then $pos(pc_n, cand_i)=0$.

For example, considering a chain having 3 positions (S=3), whose norm is of the class 5-3:

2 4 | 0 1 5 | 4 3 |

And the following candidates to be added with their scores (the repeated PCs are marked in Italics Bold):

candidate$_0$ = {***5 1 0***}
score = [(3-3) + (3-3) + (3-3)] / (3*3) = 0
candidate$_1$ = {B ***0 1*** }
score = [(3-0) + (3-3) + (3-3)] / (3*3) = 0.33…
candidate$_2$ = {***2 0*** B}
score = [(3-2) + (3-3) + (3-0)] / (3*3) = 0.44…
candidate$_3$ = {***5*** 7 8}
score = [(3-3) + (3-0) + (3-0)] / (3*3) = 0.66…
candidate$_4$ = {***2*** 6 7}
score = [(3-2) + (3-0) + (3-0)] / (3*3) = 0.77…
candidate$_5$ = {8 7 6}
score=[(3-0) + (3-0) + (3-0)] / (3*3) = 1

It can be easily seen that –according to this criterion- the 'best' candidate is scored by 1 and is also the one that adds three new PCs to the chain whilst the 'worst' candidate is scored by 0 and it merely repeats the PCs of the norm of the chain. Finally, it is worth noting that is not mandatory at all to select the candidate with the highest score, because there may be many other criteria by which a PCS may not be considered a 'good' candidate (just to mention one of them, the PCS candidate may belong to a SC that was decided to be excluded because of aural or stylistic reasons).

3.4 Constructing *chains* with more than one norm

It is possible to extend the already explained method for constructing chains to obtain a CM with different norms. A case having special relevance in music is presented here. If it is desired to achieve a CM whose vertical norm is always of the same SC, *x*, whilst all the horizontal norms are of different classes, *a*, *b*, *c*, *d* and *e* and supposing the cardinality of the norms is always 5, the scheme of the chain to be generated is shown in Table 4[12]:

a		*b*		*c*		*d*		*e*	
**	***	**	***	**	***	**	***	**	***
	x		*x*		*x*		*x*		

Table 4

That will be the base for the CM shown in Table 5 below:

(sc)	*x*	*x*	*x*	*x*	*x*
a	**	***			
b		**	***		
c			**	***	
d				**	***
e	***				**

Table 5

Achieving such structures is a key for mastering atonal counterpoint, since they may be very effectively used for controlling both the simultaneity and the succession of PCs and their SCs on a polyphonic musical thread.

3.5 Using *pcslib* in the *SuperCollider* environment to construct *chains* and CMs

In this section, a use case in which the construction of the chains and CMs above mentioned will be presented.

Being the following PCSs:

```
a = PCS('5-1');
b = PCS('5-21');
c = PCS('5-35');
d = PCS('5-7');
e = PCS('5-33');
x = PCS('5-12'); // 5-Z12
```

A chain may be constructed using the methods explained in Section 3.2. First a *PCSChain* is created and its initial norm is set. Then the candidates for continuing it are computed and evaluated, and a selected partition out of the candidates list is added:

```
~chain = PCSChain.new.norm_(a);
~chain.candidates(false);
~chain.addCand(7);
```

[11] This criterion was formalized by Pablo Di Liscia [3].

[12] Where each '*' represents a Pitch-class.

Next, the criteria described above to create a chain is applied. Note that it is known beforehand that the chain can be constructed, so just to execute the following *ad hoc* algorithm is needed (the resulting chain is show in Table 6):

```
[x, b, x, c, x, d, x, e].do({ arg pcs;
  ~chain.norm = pcs;
  ~chain.candidates(false);
  ~chain.candList.notEmpty.if({
    ~chain.addCand(
      ~chain.scores.indexOf(
        ~chain.scores.maxItem
      );
    );
  }, {
    "candidates for %"
    .format(pcs.name).throw;
  });
});
```

A (5-1)		**B (5-21)**		**C (5-35)**		**D (5-7)**		**E (5-33)**	
03	*124*	*67*	*3AB*	*14*	*68B*	*5A*	*349*	*68*	*02A*
	X (5-Z12)		**X (5-Z12)**		**X (5-Z12)**		**X (5-Z12)**		

Table 6

Now a *PCSMatrix* from the generated chain is created (shown in Table 7):

```
~matrix = PCSMatrix.fromChain(~chain);
```

set-class	**4-11**	**X (5-Z12)**	**X (5-Z12)**	**X (5-Z12)**	**X (5-Z12)**
A (5-1)	03	124			
B (5-21)		67	3AB		
C (5-35)			14	68B	
D (5-7)				5A	349
E (5-33)	02A				68

Table 7

and the default swapping algorithm is performed to improve the distribution of the CM:

```
~matrix.swapping;
```

Finally, the PC 9 of the fourth row is duplicated in the first column to keep all the vertical norms within the SC 5-Z12:

```
~matrix.addAt(3, 0, PCS[9]);
```

which will result in Table 8.

set-class	**X (5-Z12)**	**X (5-Z12)**	**X (5-Z12)**	**X (5-Z12)**	**X (5-Z12)**
A (5-1)	02	1	4		3
B (5-21)	A	7	3	B	6
C (5-35)		6	1B	8	4
D (5-7)	39	4		5A	9
E (5-33)	0	2	A	6	8

Table 8

The process described so far results in a particular and coherent PCS distribution in two dimensions but it only defines the sonic potential of the pitch organization. There are many possible 'realizations' of this structural organization which will turn in different musical results. No rhythmic constrains are given except for the vertical alignment that provides a relative temporal 'window' within which the harmony can remain in norm. Other parameters of the pitch organization like register, range and timbre are not given either. All of these basic variables remain free for further development.

4 Conclusion

The *pcslib-sc* library presented in this paper is a flexible and robust tool for effectively handling the main features of atonal pitch organization. Although the structures that can be created are highly abstract, they may constitute the basis for pitched music organization. The realization of such abstract structures (i.e., the conversion of them in music) requires the setting of numerous sound features (such as register, duration, intensity and timbre among others) which are suppose to be congruent with the underlying pitch organization. *SuperCollider* is a very powerful environment for the latter accomplishment, and the objective of the *pcslib-sc* library was to add to it yet a new extension of its capacities.

5 Acknowledgments

The authors thank to *Universidad Nacional de Quilmes* (Buenos Aires, Argentina) for supporting and hosting this research.

References

[1] Milton Babbit. 1961. *Set Structure as Compositional Determinant*. Journal of Music Theory 5, no.1, USA.

[2] Oscar Pablo Di Liscia. 2012. *PCSLIB* site. https://puredata.info/Members/pdiliscia/pcslib/

[3] Oscar Pablo Di Liscia. 2012: *PCSLIB* reference. https://puredata.info/Members/pdiliscia/pcslib/Help-English.doc/view

[4] O. P. Di Liscia and P. Cetta. 2009. *Pitch-class composition in the pd environment*. XII Simposio Internacional de Computación y Música, Recife, Brasil.

[5] O. P. Di Liscia. 2011. *Medidas de similitud entre conjuntos ordenados de grados cromáticos*. Revista de Investigación

Multimedia, Vol III, IUNA, Buenos Aires. Argentina.

[6] Allen Forte. 1974. *The Structure of Atonal Music*. Yale University Press. England.

[7] Robert Morris. 1984. *Combinatorialty without the aggregate*. Perspectives of new Music. USA.

[8] Robert Morris. 1987. *Composition with Pitch-classes: A Theory of Compositional Design*. Yale University Press. USA.

[9] Puckette, Miller. 2007. *The theory and technique of electronic music*,
world scientific publishing co. Pte. Ltd.

[10] Mccartney, James. 2002. *Rethinking the computer music language: supercollider*. Computer music journal, 26:4:61-68, mit press, massachussets.