Elements of Algorithmic Composition

Oscar Riveros

June 2, 2013

Introduccion

La composicion algorithmica es el resultado de automaizar tareas tediosas, repetitivas y complejas del proceso de composicion musical tradicional, por ejemplo supongamos tenemos un cantus firmus, y queremos encontrar un contrapunto de alguna especie, podemos hacerlo a mano, pero cabe preguntarse: ¿mi solucion o mi contrapunto es uno de cuantos posibles?, es decir, perdemos toda lariqueza del gusto artistico, por que de escribir todos eso contrapuntos a mano, no tendriamos vida, entonces ocupando tecnicas como la constraint programming, podemos modelar una Meta-Structura de nuestro contrapunto, dandole una intencion de fondo mas alla de la simple causalidad y limitaciones humanas, asi podriamos pedir por ejemplo que aparte de seguir las normas delcontrapunto estricto, que hubiera algun tipo de relacion mas fuerte entre el material generado, digamos que el climax este en cierta hubicacion o bajo ciertas proporciones, como la proporcion dorada, podriamos elegir algunas notas que representen un motivo mas generico, y el contrapunto generado finalemnte seria una interpolacion sobre nuestra boceto intencional, el unico limite en la composicion algorithmica es nuestro conocimiento.

De esta forma la composicion algorithmica se diferencia de la tradicional, en que la primera esta limitada a nuestra condicion humana, y la segunda ocurre en un plano muy distinto un plano transhumano, donde se modela una meta estrucura que en terminos tecnicos es informacion pura, que se materializa en Musica en el sentido tradicional.

En gran medida un compositor algoritmico es una fusion de un Maestro Compositor tradicional con un Experto en Ciencias dela Computacion, por lo que nuestros programas son tales en el sentido usoal, pero en vez de tener codigocompilado, tenemos Meta Musica, ya sea esta MIDI, MusicXML, LilyPond, Fomus, etc.

Las tecnicas usadas tracienden las tecnologias y los lenguajes de programacion, y la Composicion Algorithmica es una disiplina en si misma, dando origen a resultados y metdos no aplicables en la composicion tradicional.

Getting Started with Algorithmic Composition

Primero declaramos la cabecera de nuestra composicion, en la cual incluimos las librerias a usar, todo depende de lo que queramos hacer o tengamosen mente, siempre es bueno tener una idea predefinida, pero de no ser posible, es bueno probar ideas nuevas en bocetos, en algun momento del proceso, tendremosun buen material para trabajar.

Una delas cosas que quiero hacer incapie y quiero ser categorico al respecto, NUNCA USEN ALEATORIEDAD PARA SUS CMPOSICIONES, el cerebro reconoce patrones y todo en musica desde la onda basica del timbre de un instrumento son patrones, el ruido se diferencia de la musica, por que el ruido es aleatorio, y el sonido la musica o alguna estructura superios como una composicion musical, es tal y sucalidad depende, de su patron dominante, es decir podemos crear musica muy cmpleja, pero siempre ha de ser determinista, en el sentido de poder ser reconstruida, es la unica forma que el cerebro la considere una estructura superior y no somplemente un sonido ambiental agradable.

```
# palette of libraries

from IPython.external import mathjax
from sympy import *
from sympy.utilities.iterables import variations
from sympy.plotting import plot3d
from music21 import *
from itertools import cycle

x, y, z, r, s, t                    = symbols('x y z r s t')
m, n, i, j, k                       = symbols('m n i j k', integer=True)
f, g, h, eq, plus, minus, div, times = symbols('f g h eq plus minus div times', cls=Function)
```

Lo siguiente es definir el motivo con el o los motivos con que queremos trabajar, un motivo es un contorno melodico y ritmico, pero no corresponde a un grupo de notas concreto, y depende del contexto en que se desarrolla, por ejemplo el mismo motivo se comportaria diferente, en un contexto cromatico, que en uno tonal, pero el oido seria capas de reonocer ambas materializaciones del motivo como lamisma.

En nuestra implementacion del motivo, definimos primero, algunas funciones basicas, que operaran sobre las notas del entramado, que definiremos mas adelante, el entramado son los hilos con los que tejemos la musica y y el motivo es el diseño que armamos con esos hilos.

```
# motive definition

eq    = Lambda((x, y), x)
plus  = Lambda((x, y), x + y)
minus = Lambda((x, y), x - y)
times = Lambda((x, y), x * y)
div   = Lambda((x, y), x / y)

motive_male =
[(eq, 0), (plus, 2), (plus, 2), (minus, 5), (plus, 3), (plus, 6), (plus, 7), (minus, 8)]
rhythm_male =
cycle([1./2, 1./2, 1./2, 1./2, 1./2, 1./1, 1./1, 1./1] * 2)

motive_female =
[(eq, 0), (plus, 3), (plus, 6), (minus, 7), (plus, 5), (plus, 2), (plus, 5), (minus, 7)]
rhythm_female =
cycle([1./2, 1./2, 1./2, 1./2, 1./2, 1./1, 1./1, 1./1][::-1] * 2)
```

ahora que tenemos el motivo, necesitamos un expansor, algo que desarrolle el motivo sobre el entramado, seria como el cabezal de nuestra tejedora que toma nuestro diseño basico y lo extiende a lo largo de los hilos definidos por el entramado.

```
# motive expander

def framework(start, end, intervals):
    material = [start]
    tmp = start

    for index in range(start, end):
        if tmp < end:
            tmp = tmp + intervals.next()
            material = material + [tmp]

    return material


def expander(motive, length):
    return map(lambda x, y: x[0](x[1], y), motive * length, range(len(motive * length)))
```

Aqui implementamos, un creador generico de partes, dependiente de nuestra libreria musical elejida.

```
def make_part(part, material, material_maping, rhythm, rhythm_mapping, octave):

    map(part.append,
        map(lambda pc:
            note.Note(material_maping(pc),
                      quarterLength = rhythm_mapping(next(rhythm)),
                      octave = octave),
            material))

    return part
```

Ahora definimos, algunos parametros globales, por ejemplo, escalas por zonas, aqui es una implementacion muy ridimentaria, posteriormente crearemos un acon un amayor definicion y detalle, sobre el resultado final.

```
intervalic_pattern = cycle([3, 2, 2, 3, 2] * 2 + [2, 1, 2, 2, 2, 1, 2] * 2)
size               = 12

# global instruments

voice1 = stream.Part()
voice2 = stream.Part()
voice3 = stream.Part()
voice4 = stream.Part()

# global motives development

material        = framework(0, 127, intervalic_pattern)
voice1_material = expander(motive_male         , size)
voice2_material = expander(motive_female       , size)
voice3_material = expander(motive_male[::-1]   , size)
voice4_material = expander(motive_female[::-1], size)
```

Definimos caracteristicas especiales de cada una de las secciones

```
# Section A

voice1_material_maping = lambda x: list(material)[int(x**1) % len(material)]
voice2_material_maping = lambda x: list(material)[int(x**2) % len(material)]
voice3_material_maping = lambda x: list(material)[int(x**3) % len(material)]
voice4_material_maping = lambda x: list(material)[int(x**4) % len(material)]

voice1_rhythm_maping   = lambda x: float(x*1)
voice2_rhythm_maping   = lambda x: float(x*2)
voice3_rhythm_maping   = lambda x: float(x*3)
voice4_rhythm_maping   = lambda x: float(x*4)

voice1 = make_part(voice1, voice1_material, voice1_material_maping, rhythm_male, voice1_rhythm_maping, 5)
voice2 = make_part(voice2, voice2_material, voice2_material_maping, rhythm_male, voice2_rhythm_maping, 4)
voice3 = make_part(voice3, voice3_material, voice3_material_maping, rhythm_male, voice3_rhythm_maping, 3)
voice4 = make_part(voice4, voice4_material, voice4_material_maping, rhythm_male, voice4_rhythm_maping, 2)

# Section B

voice1_material_maping = lambda x: list(material)[int(x**4) % len(material)]
```

```
voice2_material_maping = lambda x: list(material)[int(x**3) % len(material)]
voice3_material_maping = lambda x: list(material)[int(x**2) % len(material)]
voice4_material_maping = lambda x: list(material)[int(x**1) % len(material)]

voice1_rhythm_maping   = lambda x: float(x*4)
voice2_rhythm_maping   = lambda x: float(x*3)
voice3_rhythm_maping   = lambda x: float(x*2)
voice4_rhythm_maping   = lambda x: float(x*1)

voice1 = make_part(voice1, voice1_material, voice1_material_maping, rhythm_male, voice1_rhythm_maping, 5)
voice2 = make_part(voice2, voice2_material, voice2_material_maping, rhythm_male, voice2_rhythm_maping, 4)
voice3 = make_part(voice3, voice3_material, voice3_material_maping, rhythm_male, voice3_rhythm_maping, 3)
voice4 = make_part(voice4, voice4_material, voice4_material_maping, rhythm_male, voice4_rhythm_maping, 2)
```

y finalmente materializamos nuestra Meta-Partitura

```
# Let's Put It All Together

new = stream.Stream()
new.insert(0, voice1)
new.insert(4, voice2)
new.insert(8, voice3)
new.insert(16,voice4)
new.show()
```