Elementos de Composición Algorítmica

Oscar Riveros

June 2, 2013

Introduccion

La composición algorítmica es el resultado de la automatización de proceso de composición musical tradicional tediosa, repetitiva y compleja, por ejemplo supongamos que tenemos un cantus firmus, y quiere encontrar un contrapunto de algún tipo, podemos hacerlo a mano, pero la pregunta es: ¿ mi solución o mi contrapunto es uno de los pocos posible?, es decir, se pierde toda la riqueza de gusto artístico, que para escribir todo lo que contrapuntos lado, no tendríamos vida, tomando técnicas como la programación con restricciones, podemos modelar una meta -estructura de nuestro contrapunto, dando un fondo intención más allá de la simple causalidad y limitaciones humanas, por lo que se puede pedir, por ejemplo, además de seguir las normas de estricto contrapunto, había una especie de fuerte relación entre el material generado, dicen que el clímax es de alguna hubicacion o bajo ciertas proporciones, ya que la proporción de oro, podríamos elegir algunas notas que representan un motivo más genérico, y, finalmente, el contrapunto generado sería una interpolación deliberada en nuestro esquema, el único límite de la composición algorítmica, es nuestro conocimiento.

Así, la composición algorítmica difiere de lo tradicional que la primera se limita a la condición humana, y el segundo se produce en un plano transhumana muy diferente que una estructura diana se modela en términos técnicos es información pura, materializada en Música, en el sentido tradicional.

En gran parte un compositor algorítmico es una fusión de un Maestro Compositor tradicional a un experto en informática, para que nuestros programas son tales en el sentido habitual, pero en lugar de código compilado, es meta-música, ya sea esta MIDI, MusicXML, LilyPond, fomus, etc

Las técnicas utilizadas trascender tecnologías y lenguajes de programación, y Composición algorítmica es un disipline sí, dando lugar a metdos resultados y no aplicable en la composición tradicional.

Comencemos

Primero declaramos la parte superior de nuestra composición, en la que se incluyen las bibliotecas de usar, todo depende de lo que hacer o tener en cuenta, siempre es bueno tener una idea preconcebida, pero si no es posible, es bueno probar nuevas ideas en los bocetos, en algún momento del proceso, tener un buen material para trabajar.

Una de las cosas que quiero hacer hincapié en y quiero ser categórico al respecto, NUNCA UTILIZA TU COPOSICIONES RANDOMNESS, el cerebro reconoce patrones y todo en la música del anillo de onda básica de un instrumento son los patrones, el ruido a diferencia de la música, que el ruido es al azar, y el sonido de la música o alguna estructura superior como una composición musical, que sólo depende de su calidad, su patrón dominante, lo que significa que puede crear música muy compleja, pero siempre será determinista, en el sentido de ser reconstruida, es la única manera de que el cerebro considera una estructura superior, y no simplemente un sonido ambiente 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 tema con el que trabajamos, un motivo es un contorno melódico y rítmico, pero no para un grupo determinado de notas, y depende del contexto en que se desarrolla, por ejemplo el mismo motivo se comportan de manera diferente, una cromática contexto, ese contexto tonal, pero el oído reconocerían ambas capas como realizaciones de un mismo motivo.

En nuestra implementación del motivo, en primer lugar definimos algunas funciones básicas que operan en las notas de la red, que definiremos más adelante, el marco son los hilos que tejen la música y el motivo es el diseño que armamos con esos temas.

```
# 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 tenemos el motivo, necesitamos un expansor, que desarrolla el motivo en la red, desea nuestra cabeza tejedora que tiene nuestro diseño básico y se extiende a lo largo de los hilos definidos por la red.

```
# 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)))
```

Aquí ponemos en práctica un fabricante de partes de genéricos, que dependen de nuestra biblioteca musical elegido.

```
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 parámetros globales, como las escalas de áreas, aquí es una implementación rudimentaria, a continuación, crear una con mayor definición y detalle, con 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 las características 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 materializar 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()
```

Full Source Code: https://github.com/maxtuno/Advanced-Algorithmic-Composition
Materialized Music: https://soundcloud.com/maxtuno/aac-ex102
Blog:
http://oscar-riveros.blogspot.com/
http://mx-clojure.blogspot.com/
Facebook Page: https://www.facebook.com/pages/Oscar-Riveros/207414032634682