AUGMENTED LIVE CODING: HARNESSING LINKED DATA IN MUSICAL PERFORMANCES

Alo Allik
Queen Mary University of London
a.allik@qmul.ac.uk

ABSTRACT

This demo explores how to augment live coding as an improvisatory musical performance practice with evolutionary sound synthesis, audio feature extraction and linked semantic data formats to aid the performer through the complexities of digital performance environments. The *evolver* programming environment uses gene expression synthesis to generate complex synthesis graphs during live coding performances. The main principle guiding the evolution is based on extracting audio features for comparison to a target sound. The audio features are also employed as an informative framework for the performer in an otherwise featureless collection of synthesizer. The structure of the evolutionary synthesis environment is published in a Semantic Web ontology. The synthesizers are stored in a database using the JSON-LD data format to enable linking the synthesizer data to the ontology and external data sources.

1. INTRODUCTION

Live coding has been firmly established as an improvisatory musical performance practice. Musicians-programmers have been exploring alternative methods to user-interface-driven production software in order to explore more spontaneous and flexible ways to compose music in real time (see for example [2] for a review of different live coding practices that existed already over a decade ago). At the same time, the advances in computing speed and power are constantly shifting the boundaries of what is possible to achieve with sound synthesis, algorithmic composition and performance interfaces in the context of real time interactive music systems. Arguably this is the main factor behind the proliferation of real time information systems used in musical performances. However, the abundance and complexity of musical algorithms constitute a specification problem for the artist, who is constantly faced with difficult choices regarding levels of granularity for musical parameter control. Writing a low level sound generating function live on stage does not necessarily make for a captivating or even intellectually invigorating experience for the audience or, arguably, the performer. At the other end of this imaginary spectrum, graphical user interfaces of proprietary music production applications enforce preconceived and often limited compositional principles upon users, seriously stifling the latent potential of algorithmic creativity. The optimal parameter control level lies somewhere in between these two extremes. This demo proposes a framework that aspires to enhance the live coding practice by evolutionary computing and linked data technologies. For sound synthesis, an evolutionary algorithm is utilised that enables evolving large populations of complex synthesis graphs, either in real time or for later reuse. The structure of the evolutionary synthesis process is described in a light-weight OWL ontology, while the graphs are stored in a CouchDB database and linked to the ontology using JSON-LD, a semantic extension of the standard JSON format.

2. GENE EXPRESSION SYNTHESIS

Gene expression synthesis (GES) [1] is a way to evolve sound synthesizers in computer code. These synthesizers are computer programs that produce sound when executed. Gene expression synthesis uses the methods of gene expression programming [3]. The computer programs evolved with GES are sound synthesis graphs in this case implemented in the SuperCollider programming environment [1]. Each solution generates a SuperCollider SynthDef object. The functions in a GES chromosome are Unit Generators, sound generating functions that serve as basic building blocks of synthesis graphs. Fitness is primarily evaluated in terms of a distance metric of features from target audio, while secondary methods are used to ensure the structural and functional integrity of each synthesis graph as well as balancing factors between resource efficiency and graph complexity. The main fitness function measures the distance of audio feature vectors between each candidate synthesizer and a target sound, which is selected by the user depending on the context of the experiment. Resource efficiency measure has been implemented in order to imitate the condition of limited resources of natural selection, so each candidate solution is assigned a CPU usage value measured during the execution of each synthesizer. To counteract a tendency towards simpler graphs as a result, a conflicting fitness pressure is introduced to encourage structural complexity in the form

[1] http://supercollider.github.io

of awarding greater nesting depth. Once each individual has been assigned a fitness value, the population is subjected to various standard genetic operators: replication, mutation, transposition and recombination.

3. LINKING SYNTHESIZER DATA

Due to large volumes of data potentially involving thousands of candidate solutions deemed suitable for performances, GES synthesizers with their associated metadata are stored in a CouchDB[2] database as JSON data structures. This includes all the data necessary to reconstruct each synthesizer for use during a performance or as sources for subsequent evolutionary synthesis experiments. This demo is implemented in a live coding environment that integrates a customised SuperCollider system with a lightweight OWL ontology[3] that enables representation of GES synthesizers on the Semantic Web and linking of the synthesizer audio features to the concepts defined in the Audio Feature Ontology framework[4]. The synthesizer data that is communicated between CouchDB and SuperCollider is expressed in terms of the GES ontology using JSON-LD [4]. Listing 1 shows a fragment of a GES synthesizer data structure. The embedded context defines the ontology namespace so that the GES ontology classes and properties can be meaningfully expressed in JSON enabling linking concepts from external sources. In this example, the GES ontology defines Spectral Centroid and Spectral Flatness audio features and links to the Audio Feature Vocabulary, so it becomes possible to query more information about these features, for example, finding information about how these particular features are computed by querying steps of computational workflows.

```
{
   "@context": {
      "ges": "http://geen.tehis.net/ontology/"
   },
   "_id": "1a75f4f87bdcac24b6ba5fc25c003ab2",
   "@type": "ges:Synth",
   "ges:environment": {
      "@type": "ges:Environment",
      "ges:headsize": 24,
      "ges:numgenes": 1,
      "ges:linker": {
         "@type": "ges:Function",
         "ges:name": "*",
         "ges:class": "AbstractFunction"
      }
   },
   "ges:defname": "gep_gen000_061_141212_225728",
   "ges:genome": [ "LFPar", "LFGauss", "SinOsc",
      "v", "PMOsc", "e", "j", "a", "c", "a"
   ],
   "ges:features": {
      "ges:centroid": {
         "ges:mean": 526.07188020057,
         "ges:std_dev": 161.03829149232
      },
      "ges:flatness": {
         "ges:mean": 0.032055785092016,
         "ges:std_dev": 0.017184103858522
      }
   }
}
```

Listing 1. Fragment of a GES synthesizer JSON structure

4. THE EVOLVER ENVIRONMENT

The gene expression synthesis algorithm can be used for isolated experiments to generate populations of synthesizers for reuse during performances. However, the evolver environment enables the performer to evolve and play synthesizers live on stage. The semantically enriched live coding environment aids the performer in the decision making process when selecting synthesizers from the database or evolving them in real time. The synthesizers are classified according to audio feature vectors and different maps are created to visualize the distribution of feature vectors from different perspectives. For example, a plot of spectral flatness - how noise-like a signal is - against spectral centroid - how bright the sound is - gives an idea about the characteristic of each synthesizer. This can be further aided by classifying synthesizers with a self-organised 2-dimensional map of MFCC vectors. The performer can make informed selections of synthesizers either for use in the real time live coding composition process or as source material for real time evolution by going through the iterations of the GES algorithm. One of the performance strategies involves selecting 2 chromosomes from the database to serve as source material for on-stage experiments. These 2 chromosomes are then subjected to genetic operators, including recombination, which involves creating 2 new individuals by exchanging genetic material between the initial 2 chromosomes. The population can be subsequently grown by doubling the number of individuals each generation, evaluating the fitness of each, classifying them using the features and the existing data and selecting which ones to use in the performance based on this information. The evolved synthesizers can be used in a number of different ways as compositional elements. The target sound towards which the algorithm is converging is selected according to context. For example, percussive sounds are more suitable if the GES synthesizers are used in a dance music context to fill rhythm patterns, whereas continuous drone-like sounds are more suitable for building ambient soundscapes.

[2] http://couchdb.apache.org
[3] http://geen.tehis.net/ontology
[4] http://sovarr.c4dm.eecs.qmul.ac.uk

5. REFERENCES

[1] A. Allik. Gene expression synthesis. In *Proceedings of the ICMC/SMC, Athens, 14-19 September*, 2014.

[2] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(3):321–330, 2003.

[3] C. Ferreira. Gene expression programming: A new adaptive algorithm for solving problems. *Complex Systems*, 13(2):87–129, 2001.

[4] M. Lanthaler and C. Gütl. On using JSON-LD to create evolvable restful services. In *Proceedings of the 3rd International Workshop on RESTful Design at WWW2012*, 2012.