An Approach to Live Algorithmic Composition using Conductive

Renick Bell

May 9, 2013

2 intro

Renick Bell

- doctoral student at Tama Art University in Tokyo
- working on

Conductive, a Haskell library for live coding

- presented a paper about Conductive at LAC 2011 in Maynooth

photo of the Tama Art University library, http://openbuildings.com/buildings/tama-art-university-library-profile-148/media

3 outline

1. intention
2. Conductive concepts
3. problem with previous performance style
4. generating interonset interval (IOI) patterns
5. density
6. TimespanMap
7. evaluation
8. future

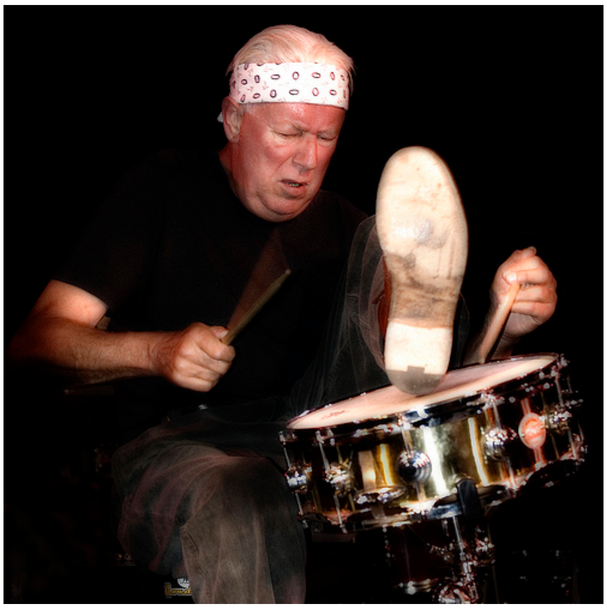
4 intention

- to **perform generative music live** with the computer as an active partner
- to make it possible for me to **do much more than I am physically able to do** on traditional musical instruments

5 not just Han Bennink

I showed this slide in my presentation in 2011.

Han Bennink playing with Peter Brötzmann at the RPI Armory, Troy, NY (25 April 2008); photo by Mark A. Lunt, Creative Commons license; http://www.flickr.com/photos/maldoror-2008/2442869131/in/photostream/



6 sometimes I want Paal Nilssen-Love

He might not play the wall or his

mouth, but he sometimes plays more furiously.

Paal Nilssen-Love @ Smeltehytta, Kongsberg Jazz Festival 2011-07-09, photo by Heiko Purnhagen (c) 2012

Actually, though, a Hans Bennik or Paul Nillsen-Love is not enough.

7 actually an orchestra of Han Benninks and Paul Nilssen-Loves

This is actually what I want: Butch Morris conducting an ensemble of Han Benninks and Paal Nilssen-Loves.

But how can I get there?

photo of Butch Morris by Claudio Casanova
http://outlawpoetry.com/2013/02/12/steve-dalachinsky-conduction-is/

8 background

the Conductive library:

- facilitates live coding in Haskell
- for control data, using other synthesizers to synthesize sound
- first release in Nov. 2010

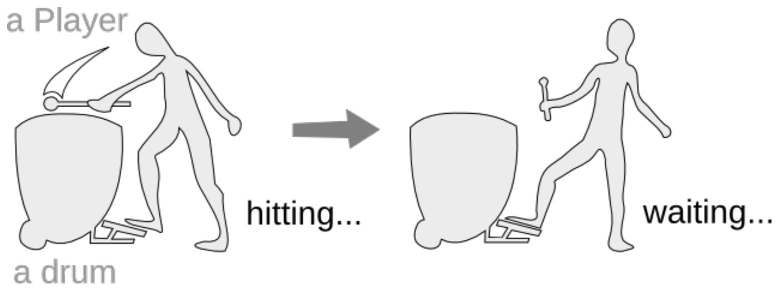
9 Conductive concepts

- Player
- interonset interval (IOI) function
- action function
- TempoClock

10 Player

A Player is an **abstraction** that:

- **initiates a process**
- **waits for a period of time**
- initiates another process
- waits for another period of time
- and **repeats** this as long as it is running.



The processes that a Player initiates are described in action functions.

In my case at present, the action functions are triggering sample-playing synthdefs in scsynth (SuperCollider synthesis engine).

The focus at the moment is on how long to wait, which determines the rhythm of the music.

11 IOI

IOI = interonset interval

For one Player, **the period from the start of one event to the start of the next event**.

It is possible than an event has not finished running when the next event starts.

It is **expressed in terms of beats**, which in turn are based on a TempoClock.

12 IOI function (1)

An IOI function is passed:

- the MusicalEnvironment
- the Player
- the beat that it should be playing on
- and the actual current beat.

Using this data or not, it calculates **the next beat** that it should run its action function on.

13 IOI function (2)

They could be based on lists of IOI values, like in a typical score.

[1,1,1,1] might represent a measure of four beats.

A simple technique is looping through a list, succesively returning each value.

The design is up to the user.

14 constraints of live coding: time/concentration

When performing, **I don't have enough time to focus on each aspect deeply**.

There's no time to write many lists of IOI values by hand.

15 generating sets of IOI values

I'm currently using an algorithm like this:

1. **make a set of potential IOIs based on a core value**
2. **make a set of subphrases based on those potential IOIs**
3. **make a final phrase by choosing from those subphrases up to a user-specified length of time**

This uses the concepts of **repetition** and **variation** to create arguably aesthetically-pleasing patterns.

16 example of a generated set

This example leaves out some of the parameters and details; for a complete example, see the paper.

- **core unit = 1**
- **potential IOIs = 1, 2, 3**
- **potential subphrases = [1,2,1], [3,1], [2,2]**
- **final phrase = [1,2,1,1,2,1,2,2,3,1]**

This final phrase could be called a pattern.

17 what is the relationship between lists?

Consider two sets of patterns.

```
1. [1,1,1,1]
2. [1.5,1,0.25,0.25,3]
```

1 and 2 are **dissimilar and feel unrelated**.

```
3. [1,1,1,1]
4. [1,1,1,0.5,0.5]
```

3 and 4 are **similar and feel related**.

18 livecoding, time constraints

In this version of Conductive, I often use **five to eight Players, meaning five to eight IOI patterns**.

If I **want to rapidly make series of changes** to parameters involved in performance, there's no time to think about and then write out related patterns.

One of my stated goals at the end of my presentation in 2011 was **to reduce the number of parameters** which I had to directly manipulate by **making abstractions based on musical features**.

The first of these that I have tried is **event density**.

19 density

A musical feature describing **the number of events in a given time period**.

3. [1,1,1,1] (four events in four beats, **less dense**)
4. [1,1,1,0.5,0.5] (five events in four beats, **more dense**)

20 density map

A density map is:

- based on a single IOI pattern
- a complete set of related patterns
- ordered from minimum to maximum density

21 density algorithm description

Given an IOI pattern:

- reduce the density by joining two values
- increase the density by replacing a value with:
    1. *a smaller potential IOI value*
    2. *and the difference between it and the value being replaced*
- join them into a list ordered by density

To create the density map, it is necessary to know:

- the potential IOI values
- and an input phrase.

22 example density map 1

- input phrase: 3, 1, 2, 2
- potential IOIs: 1, 2, 3

23 example density map 2

- input phrase: 3, 1, 2, 2
- potential IOIs: 1, 2, 3

a complete table:

```
0: [8]
1: [4,4]
2: [4,2,2]
3: [3,1,2,2]
4: [2,1,1,2,2]
5: [2,1,1,2,1,1]
6: [2,1,1,1,1,1,1]
7: [1,1,1,1,1,1,1,1]
```

Knowing which set of IOI values to use depends on the density value.

24 density map to IOI value

The IOI function can:

1. **refer to a density value**
2. **find the current set of IOIs**
3. **use that to determine the beat of the next event.**

25 changing the density by hand takes time

However, if there are five to eight Players, it is troublesome to constantly change the density value of each of the Players over time.

There **seems not to be enough time** to do so **and continue to focus** on the other elements of a performance.

26 TimespanMaps

TimespanMaps are structures which **deal with values that change over time**.

When passed a **time**, they return a **value**.

It could be called an "interval map".

27 how timespan maps work, timespan map algorithm

A TimespanMap contains:

- **a list of interval starting times**
- **a value for each interval**
- **a length in terms of time that determines the end of the final interval.**

When passed a time that falls within an interval, the map returns the corresponding value.

Once passed a time that exceeds the length of the TimespanMap, it loops back to the beginning of the TimespanMap.

28 example TimespanMap (1)

A very simple TimespanMap:

```
length: 2
0: "a"
1: "b"
```

29 example TimespanMap (2)

```
length: 2
0: "a"
1: "b"
```

At time 0, the value is "a". More times and their values:

- 0.5, "a"
- 1, "b"
- 1.25, "b"
- 1.99, "b"
- 2, "a"
- 3, "b"
- ...

30 writing timespan maps takes time

However, composing these TimespanMaps also takes time, particularly for writing those with a large number of intervals.

That occurs for example in the case of a value that changes gradually from one extreme to another over a relatively long period of time.

31 interpolation of timespan maps

One solution to writing such TimespanMaps is using interpolation.

At the moment, only linear interpolation is used.

Example of an interpolated TimespanMap:

```
*TestingConductive> interpolatedTimespanMap 4 2 [(0,0),(2,1)]
TimespanMap {mapLength = 2.0, timespanMap = fromList [(0.0,0.0),(0.5,0.25),(1.0,0.5),(1.5,0.75)]}
```

32 demonstration of all together

Three Players:

- kick samples
- snare samples
- hihat samples

33 the good

Assigning samples by using TimespanMaps reduced the number of Players.

Varying rhythm became much easier.

It became easier to control performances.

The results became less random and more musical.

Performances are more interesting than previously.

34 the bad

The musical feature of density is not enough.

Using only sample playback is not satisfying.

A higher-yet-still-flexible level of rhythmic specification might still be possible.

The amount of data and mutable data storage needed for a performance increased.

That complexity can also be hard to keep in one's head and manage.

35 the future

- more

  sophisticated generation of rhythms (can anyone suggest particular approaches or composers to look at?)
- more types of synthesis that are manageable live
- similar development for pitch and harmony
- development of other musical features beyond density
- concise interpolation strategies for TimespanMaps
- visualization
- code management tools for live coding
- cleaning up the code again

- and on and on...

36 performance

I am performing bass music with this system on Saturday evening.

I hope you will come to that performance.

37 more info, code

http://renickbell.net

38 thanks

Thanks to **Henning Thielemann** for discussions.

Thanks to **the reviewers** for useful suggestions on the paper.

Thanks also goes to my advisors **Akihiro Kubota** and **Yoshiharu Hamada** and **other staff at Tama Art University** for research support.

39 questions? suggestions?

- Things you don't understand? Criticisms? Comments?
- Can you recommend people with concise algorithms for generating music?
- Can you email me concise periodic or almost periodic functions with interesting properties?
- What about good books or essays that explain such functions in a friendly (or not-so-friendly) manner?