Music and Unconventional Computing

Eduardo Miranda and Andy Adamatzky (editors)

AISB Convention 2013 • University of Exeter • 3rd–5th April, 2013

Foreword from the Convention Chairs

This volume forms the proceedings of one of eight co-located symposia held at the AISB Convention 2013 that took place 3rd-5th April 2013 at the University of Exeter, UK. The convention consisted of these symposia together in four parallel tracks with five plenary talks; all papers other than the plenaries were given as talks within the symposia. This symposium-based format, which has been the standard for AISB conventions for many years, encourages collaboration and discussion among a wide variety of disciplines. Although each symposium is self contained, the convention as a whole represents a diverse array of topics from philosophy, psychology, computer science and cognitive science under the common umbrella of artificial intelligence and the simulation of behaviour.

We would like to thank the symposium organisers and their programme committees for their hard work in publicising their symposium, attracting and reviewing submissions and compiling this volume. Without these interesting, high quality symposia the convention would not be possible.

Dr Ed Keedwell & Prof. Richard Everson
AISB 2013 Convention Chairs

Published by
The Society for the Study of Artificial Intelligence and the Simulation of Behaviour
http://www.aisb.org.uk

ISBN: 978-1-908187-34-5

Table of contents:

AISB 2013 Symposium on Music and Unconventional Computing

Foreword from the symposium organisers

Research in unconventional, or nature-inspired, computing aims to uncover novel principles of efficient information processing and computation in physical, chemical and biological systems, to develop novel non-standard algorithms and computing architectures, and also to implement conventional algorithms in non-silicon, or wet, substrates.

Computers have been programmed to produce sounds as early as the beginning of the 1950's. Nowadays, the computer is ubiquitous in many aspects of music, ranging from software for musical composition and production, to systems for distribution of music on the Internet. Therefore, it is likely that future developments in Computer Science will have an impact in music technology.

This workshop will discuss ways in which unconventional modes of computation may provide new directions for future developments in Computer Music.

The seven accepted papers included in these proceedings provide an excellent glimpse of this emerging field and it is hoped that by collecting them together here we will encourage further research in this exciting new field.

Organisation:

Chairs:

Prof Eduardo Miranda, Interdisciplinary Centre for Computer Music Research (ICCMR), Plymouth University.

Prof Andy Adamatzky, Unconventional Computing Centre, University of the West of England, Bristol.

Programme Committee:

Martyn Amos, Manchester Metropolitan University, UK
Peter Beyls, St Lukas University College of Art and Design, Belgium
Tim Blackwell, Goldsmith College, UK
Cristian S. Calude, University of Auckland, New Zealand
Alexis Kirke, University of Plymouth, UK
Duncan Williams, University of Plymouth, UK
Jon McCormack, Monash University, Australia
Jonathan W. Mills, Indiana University, USA
Vladimir Privman, Clarkson University, USA
Susan Stepney, University of York, UK
Christof Teuscher, Portland State University, USA

Musical Synthesis by means of Cellular Automata and Gestalt Patterns

Luca Danieli

School of Electronic Music, Conservatory “C. Pollini” of Padua,
Via Eremitani 18, 35121, Padova, IT.

dnllcu@gmail.com

Abstract In order to emulate a composer-based approach to musical writing, the paper aims to to create of a new musical synthesis model based on gestalt matrices.
The approach is described in three different levels:
- a sound model: all instruments have their own interface and have specific roles within a musical composition. Every instrument typology is unique, and composers write for specific instrumental characteristics;
- an action model: every performer has different ways to interact with the instrument itself. Each instrument is a powerful sound provider. The action performed on the instrument has the function of “choosing” which particular sound model to play;
- a score model: every musical staff has specific functions in the score and represents a certain contribution to the outcome which is related to macro-structure analysis.

A self-generating matrix, described by use of patterns, is applied to these basic concepts, thus giving them personal stochastic behaviors and creating a comparison system to reduce data processing.

Keywords Perception, cognition, cellular automata, musical writing, gestalt patterns, emotional model.

1 Introduction

This paper presents a new sound synthesis approach based on cellular automata (CA) application.

Since the creation of the famous G*ame of life* by John Conway [1], interest on this topic has grown, finding applications in many fields of the scientific research. Conway's work consisted in two-state cells (on/off) relating to each other through a set of rules previously conceived.

The reason behind the popularity of CA can be traced to their simplicity, and to the enormous potential they hold in modeling complex systems, in spite of their simplicity [2].

Furthermore, A. Smith underlined how CA algorithms are efficient in self-reproduction, showing how the integration of different parallel-working microautomata is possible in order to create a larger macroautomaton, improving the general behavior of the system [3].

More complex CA types have been created, like the Banks' 4 states per cell [4], Codd's 8 states per cell [5] and Von Neumann's 29 states per cell [1], [3].

In other fields of scientific research, differently from the models above, the simplest two state per cell type has been implemented in a more complex structure to enlarge the variety of the outcome, as for Celerina [6]. This Cellular Automaton uses a recursive comparison, at the position t-1, between a CA array and a chosen pattern of 5 cells, thereby changing continually the state of the cell at the position t. A similar comparative approach was already faced also by Stephen Wolfram [7].

The model proposed in this paper, to enrich both the outcome and the execution of the instrument in electronic musical performance, is based on the combination of different parallel-working CA. A basic self-generating CA sound, obtained by use of mathematical series, is lead by controlling CA patterns.

Many researches have been pursued to make CA computing models more practically oriented. To achieve this goal, researchers should be able to predict the global behaviour from the local CA rules. Once this goal is achieved, one should be able to design the local rules/initial conditions from a given prescribed global behavior [2].

Here the controllers are supposed to be simple recursive patterns, acting on the spectrum as self-generating matrices, to reduce the computing and to simplify the implementation of the algorithm. These controllers represent the main four envelopes used in musical writing: crescendo, diminuendo, tenuto and the impulse function. Theoretically, since it is possible to predict the CA behavior, it is possible to foresee also the general development of a system integrating more CA algorithms.

2 Background Considerations

Instruments have their own characteristics that are perceived by people as “invariants” and related to particular *functions to music*. This music constitutes an ideal. It does not really exist in the interaction with the player, but the latter has an image of it, and through this he can relate with the instrument in a personal way called *interpretation*. It is important underlining this because we can thus define an essential characteristic for our model: the physical (usual) behavior of an instrument is different from the person's use of it. In particular the contemporary composition was characterized by strict studies on instrumental expressiveness, searching for ways to use instruments in order to produce new sounds.

If we have a piano, its *function to music* will be, for example, the physical behavior of the vibrating strings (particularly, for a composer, it focuses on the “modifications” of the sound caused by the vibrating strings) and once this relationship is defined, a person is able to imagine what interruptions, modifications, and deformations on this function would produce. The main observation remains that manipulations do not change the idea the user has of the instrument, and his method of interacting changes consequentially to what the action produces on the instrumental function. In other words, when the user has an idea of what his action will produce on the sound, he can choose his own method of interaction.

So we have to divide the synthesis in two different parameters: function and action, which must remain related to each other. For this purpose we will make use of two principles: physical modeling synthesis and cellular automata.

Physical modeling synthesis is a synthesis technique which uses mathematical models in order to synthesize the physical behavior of an instrument by the description of its physical properties. The method has shown its power in naturalness and precision in sound reproduction despite of a relevant heaviness in computing.

Most attempts to synthesize sounds based on a physical model have been based on numerical integration of the *wave equation* [8]. A general wave equation for an ideal vibrating string is given by a second order derivative equation:

$$Ky'' = e\ddot{y} \tag{1.1}$$

where K is the string tension, y is the string displacement, e is the linear mass density, y'' is the second order derivative in the x axis, and $\ddot{y}$ is the second order derivative in the *time* axis [9].

The problem for live musical performance applications, other than the high computational requirements, is that *"gestures"* which are implemented in the vibration model equation force the interaction in a gesture/gesture-vibration chain. This means that the model based on the sub-actions' summation is avoided in favor of a finished execution pattern. Another problem is that computer-based sounds in most of the cases sound artificial [9]. So, in order to solve these obstacles, it is important to find a method to relate these vibration models to a re-framed action model.

3 Sound, Action and Score patterns

A technique is given by cellular automata based on recursive patterns.

An instrument is normally a complex body determined by its resonating properties. For example, if we consider a piano, the whole spectrum is given by the resonator and the relations arising between the various vibrating strings (sympathetic resonances, beats, etc). Once the sound model is defined (strings, membranes, electronic synthesized sounds), it is possible applying a recursive pattern to it, which will enhance the stochastic behavior as if the instrument was "alive".

We can so relate spectrum portions or particular frequencies by means of simple recursive patterns. Previous studies [2] on CA show that these patterns can be traced back to the function typologies they refer to (such as the impulse functions, crescendo and diminuendo envelopes), thus creating relationships between different patterns and allowing their comparison during the musical structuring process.

For example, considering a FFT of a complex sound, the spectrum, multiplied by the same-length array filled of ones, gives simply the spectrum itself. But, if we take a matrix sampled with frequencies on the y-axis and time on the x-axis and we change the various amplitudes during the synthesis, we will obtain modulations of the original sound.

Knowing the behavior of the vehicle (e.g. the specific string proprieties) and the relationship between its various frequencies, we can create recursive patterns which relate these frequencies to each other.

If the pattern describing the harmonic succession *f, 2f, 3f, 4f, ...,* is applied to all the partials of a chosen frequency *f*, the resulting amplitudes *a* are given by:

$$a(nf) = \sum_{k=1}^{K=n} a(nf)*a(nf/k) \qquad \text{for } n > 0,\ n \text{ integer} \tag{1.2}$$

where n is the factor representing the ratio between the partial and fundamental frequency.

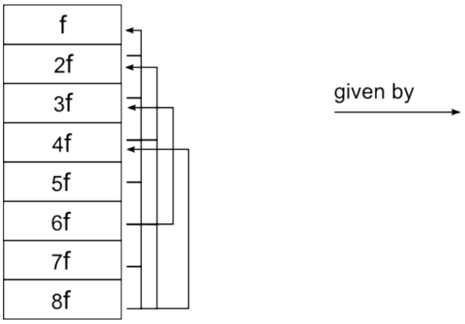


Figure 1: Frequency relationships in array form.

So the amplitude of the frequency *8f,* in Figure 1, depends by different weights of *f, 2f, 4f.*

If these frequencies change, the relative derivation *8f* changes its behavior in relation both to its own movement and to the functions describing its relation with the moving frequencies all around like in Figure 2.

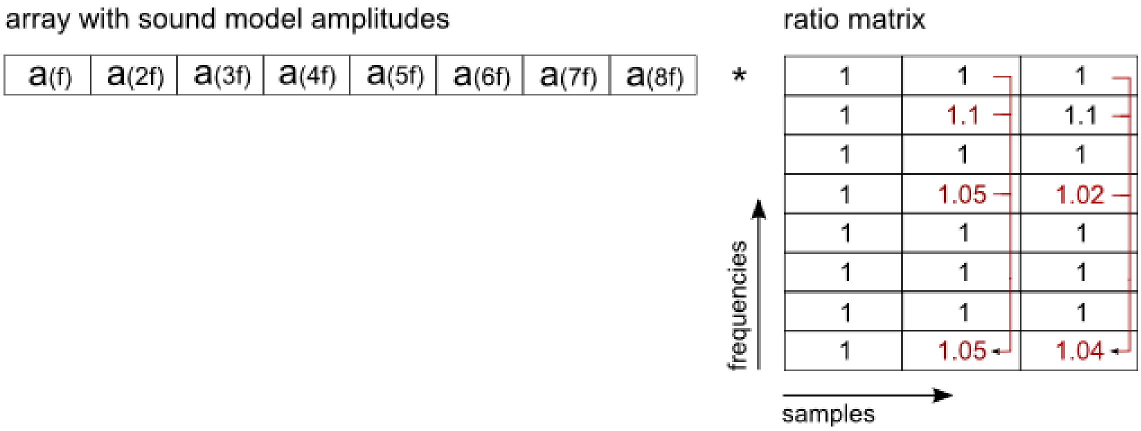


Figure 2: Matrix model for amplitude modulation.

In case we want to avoid a numerical approach and get closer to a *gestalt* representation, which is more flexible, the ratio matrix can be transformed in the *gestalt matrix,* by increasing and decreasing symbols:

$$\alpha(s, nf) = \prod_{i=1}^{I=3} \prod_{k=1}^{K=n} \alpha(s-i, nf)*\alpha(s-i, nf/k) \qquad \text{for } n > 0,\ n, s \text{ integers} \qquad (1.3)$$

where α represents the difference between the amplitudes in *a(s, nf)* and *a(s*-1*, nf)* and *s* is the sample variable. A schematic representation is in Figure 3:

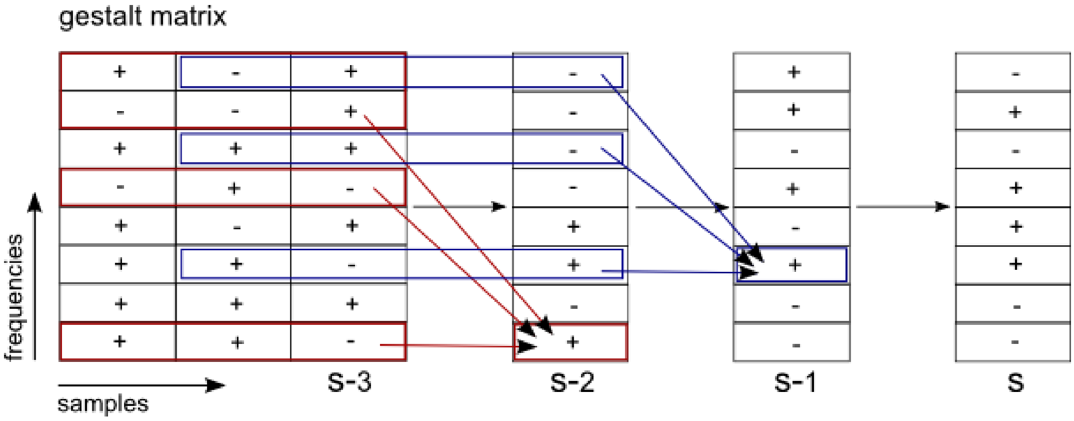


Figure 3: Gestalt matrix representation.

Here we can see how modifications in *2f* are followed by changes in *4f* and *8f* in the later control sample.

It is possible to search for patterns which lead movements outside the harmonic series, relating them to the surrounding frequency values.

The grouping in families with simpler description takes the name of *sound patterns*[¶]. This lets us describe in a synthetic form the development of the sound used in the time domain and consider its properties to influence the use and choice of particular sounds during the performance.

The computation, in real-time synthesis, is between the sound model (spectrum) and the first incoming array (the column s-5 in Figure 3), which lightens the computational process and lets the following patterns to be "initialized" after that the current array is processed.

Given the *sound patterns* describing sound models, it is simple to create *action patterns*

[¶] We will refer by means of the word sound pattern, at risk of confusing the theoretical explanation of the model, to both the simpler family description and the complex family subtrees, because what is interesting for us is their reflection of the ideal approach to their function to music (impulse, flat or decreasing envelope, …), common, by choice, to all the subtrees associated.

which lead the self-generating *sound patterns* in choosing which sound typology has to be extracted from the potential set of the instrument used. And this is possible because a sound model (e.g. a guitar) can be played in many different ways. It can be left vibrating or it can be stopped. The latter choice should be thought as an action simulating an impulsive movement with a certain duration, rather than a rest following a note. The matrix model is particularly useful because of the fast computation and the preexisting theories on matrices, which help the development of the system.

If we replace a pattern as the following:

$$\alpha(s, nf) = \sum_{i=1}^{I=3} \sum_{k=1}^{K=n} \alpha(s-i, nf)+\alpha(s-i, nf/k) \qquad \text{for } n > 0,\ n,\ s \text{ integers} \qquad (1.4)$$

with these proprieties: *“+” + “-” = 0, “+” + “+” = “+2”, “2” * ”+” = “+2”*

if $\alpha(s, nf) <=$ *“+”* -----> $\alpha(s, nf) =$ *“-”*

the *sound action* will tend generally to decrease. We can then select a decreasing ratio using different matrix lengths or even simple scalars .

An impulsive *action pattern* can be written as the following:

$$\alpha(s, nf) = \overline{\alpha(s-9, f)}$$

with the original *gestalt matrix* in Figure 4.

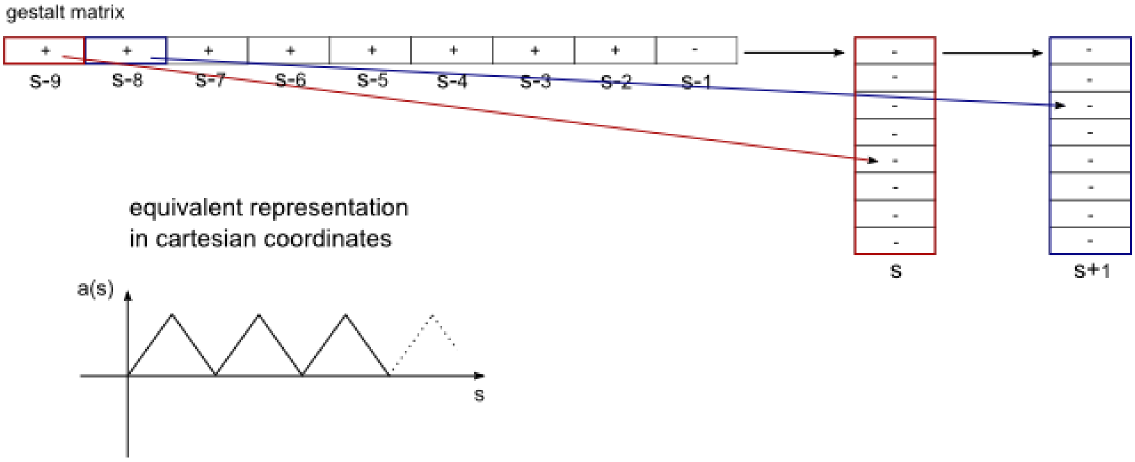


Figure 4: Impulsive gestalt matrix representation.

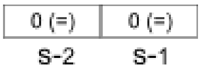
In general an *impulsive pattern* is given by:

$$\alpha(s, nf) = k * \overline{\alpha(s-i, f)} \qquad (1.5)$$

where k is the scalar determining the gradient and i is the number of columns of the original *gestalt matrix* chosen. Modifications of these parameters will give various behaviors in similar actions.

This model permits simplifications in musical writing. A composer normally relates to the principles analyzed before: *function* of an instrument *to music* and *action* of the player.

Let's make an example. A violin has commonly two *functions to music* (bowed or plucked). Regarding the bow, ideally its *function* should be represented like in Figure 5.



$$\alpha(s, nf) = \alpha(s\text{-}2, f) \qquad (1.6)$$

Figure 5: String's bow function.

But how to consider a violin playing an eight note in offbeat position on the score?

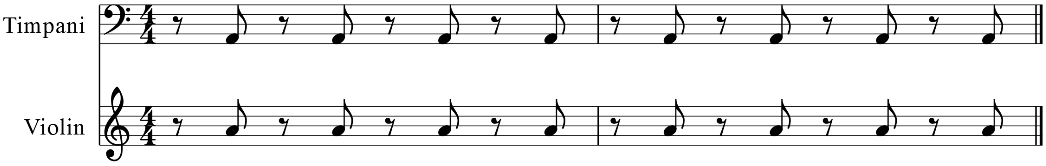


Figure 6: Score example no. 1.

Is it regarded, by the composer, as an impulse or as a tied sound? And, in Figure 7, are the timpani and violin staffs thought as impulses or as long sounds?:

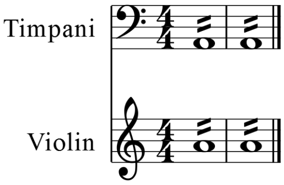


Figure 7: Score example no. 2.

In order to solve this problem it is necessary to introduce another stream of patterns: the *score patterns* which have the same properties of the previous ones and describe musical actions in the score.

Musical actions are substantially four: *staccatos, legatos, crescendo, diminuendo*.

Since the approach in Figures 4 and 5, the four actions can be described like these:

Staccato:	+	-	$\alpha(s, nf) = \alpha(s-2, f)$	(1.7)
Legato:	0 (=)	0 (=)	$\alpha(s, nf) = \alpha(s-2, f)$	(1.8)
Crescendo:	+	+	$\alpha(s, nf) = \alpha(s-2, f)$	(1.9)
Diminuendo:	-	-	$\alpha(s, nf) = \alpha(s-2, f)$	(1.10)

Figure 8: Primary musical actions table.

The basic idea is that each of these actions can make use of a different instrumental function, but keeping the same musical approach.

Therefore, the violin line in Figure 6 can be thought as a succession of impulses creating a musical sustain, which correlate it with the timpani score. Instead, in Figure 7, the timpani has a legato *musical function*¶, but of course, for the nature of the instrument, it has to be led back to a sum of impulses. In Figure 6 also, in the violin session, the musical action can use different instrumental functions depending from which the composer requires (e.g. pizzicato or bowed).

Musical Actions are used usually in conducting. In fact, often, specific gestures activate specific patterns without the direct modification on the sound. A pattern can be activated and executed continuously until the presentation of a new gesture. An example could be the suspension of a whole orchestra, where the sound decays according to the room

---

¶ We want to clarify the use of “function to music” instead of “musical function”. “Function to music” is the ideal approach a performer has on the “message” he wants to transmit. More concretely, it can be seen as the sound potential of an instrument in creating different sounds. “Musical function” instead will be used in relation to the score writing and it has the meaning of reflecting compositional intentions. Also here, it can be seen as all the instructions in order to bring out a particular sound from the whole set given by the instrument. Thus, the composer uses “musical functions” in order to create or to research well-defined “functions to music”.

properties. Here a gap arises between this gesture and the following hypothetical position. In other cases, specific gestures give directions to a small part of the orchestra performing, to move to another orchestra's session later on.

The question is: how a specific musical action has to be chosen? In this proposal, the best solution to this question is pursued by comparison of different *gestalt patterns*.

In chapter 6 we discuss, with the use of generative models, the properties of this proposal when applied to automatic score generation.

4 An aesthetic model

In this chapter we take into consideration the creation, by use of *gestalt patterns*, of an aesthetic algorithm to direct the writing process, by referring to the four main properties of a sound: attack, sustain, release and timbre.

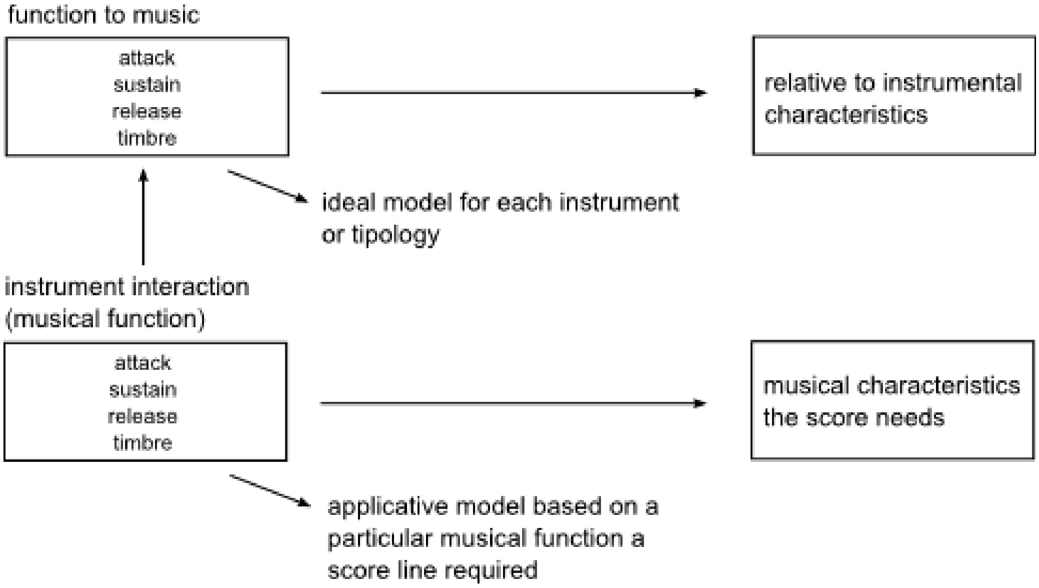

function to music
attack
sustain
release
timbre
relative to instrumental characteristics
ideal model for each instrument or tipology
instrument interaction (musical function)
attack
sustain
release
timbre
musical characteristics the score needs
applicative model based on a particular musical function a score line required

Figure 9: Instrument description model.

Since we established that f*unctions to music* are produced by the use of modern physical modeling algorithms, their description cannot be given only by *gestalt patterns*. If it's true that the *sound pattern* describing a plucked violin is not far from the impulse representation, it's also true that this gives no information about its timbre. But probably it can give information about the evolution of its spectrum or its general development in time.

The *function to music* has to answer these questions: how a particular instrument sounds? What is its behavior in time? In synthesis: how to use it and when (qualitative).

The *musical function* instead, must answer these other questions: what sound should be used in a particular score fragment? Has it to be stopped or to be tied to other notes? In synthesis: when to use it and why (quantitative).

Therefore, we have to create a comparative interface by means of *feature extraction* algorithms.

Defined the *action* concept and assuming that each performer plays as he prefers, the musical outcome should be ideally unique but shared between performers. Through the use of sound analysis algorithms it is possible to determine and, sometimes, to foresee the contribution of each instrument to the overall musical execution. The purpose is of giving feedback about the right execution of the forming score, as the following example. There are four musical staffs to generate (for example our violin and timpani staffs). Through comparison between the *sound pattern* and *sound model* some *functions to music* are chosen automatically. When a *function to music* shows a *crescendo pattern* the total spectrum of the musical fragment is little under the maximum threshold permitted, the algorithm is able to predict the crossing of the threshold chosen previously. Through a comparison between the *instrumental model* and the performing *feature extraction algorithm* the aesthetic algorithm proposed is able to adjust or to modify the musical development by changing the patterns used. Knowing the *instrumental model* selected and their singular contribution on the total spectrum, the model should be able to choose if to modify some or all of the instrumental parts of the score. In this way, we create an "alert" which tells when a musical line has to be modified.

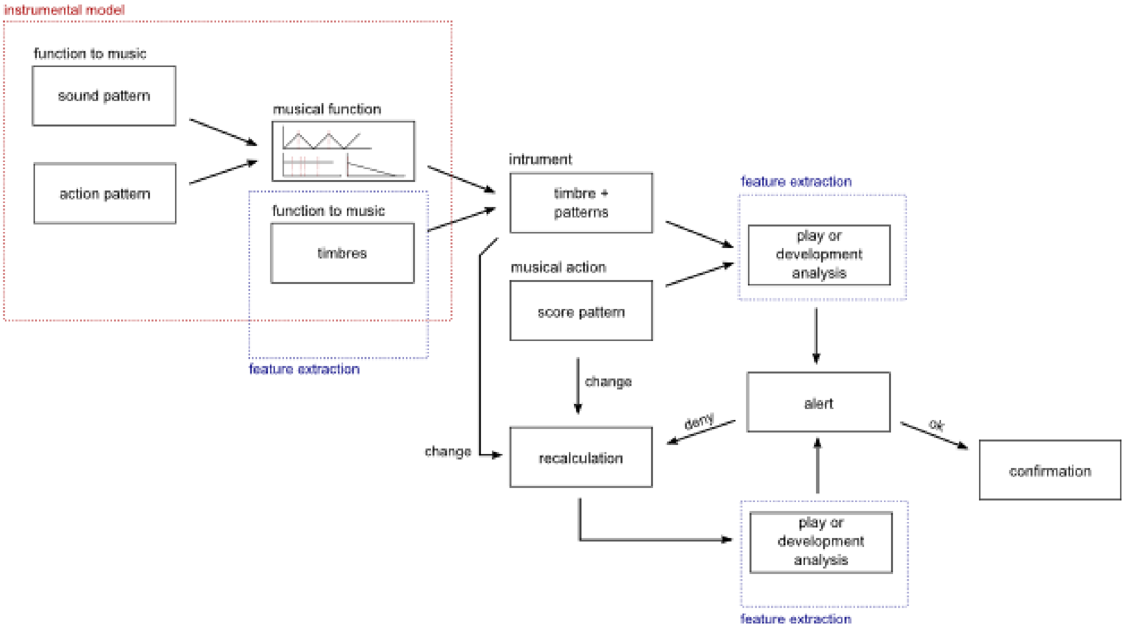


Figure 10: Aesthetic model.

In case the violin is playing *pp* and the timpani *musical action* is similar to Figure 5, the suggested amplitude for the timpani line is *pp.* Regarding a timpani musical action similar to Figure 4, the system could, instead, interpret it like a *ff.* Similar behaviors can be obtained through the substitution of *musical actions* with *musical functions*.

5 Structures by means of Data-Oriented Parsing model

We have seen in chapter 2 how to compare musical functions thus choosing what characteristics the instrument needs in order to be implemented in the score. But what is the proceeding of the structuring process? How are instruments coupled up?

To solve this problem we advert to generative grammar theories.

We will face at first the structuring topic based on the methodology approach of the DOP (Data-Oriented Parsing) model. A DOP model is a tree model extracting subtrees from a given treebank and combining subtrees of arbitrary size [10].

As Rens Bod writes [10]:

> "The phenomenon that the same input may be assigned different structural organizations is known as the *ambiguity problem*. This problem is one of the hardest problems in modeling human perception. Even in language, where a phrase-structure grammar may specify which words can be combined into constituents, the ambiguity problem is notoriously hard (cf. Manning & Schütze, 1999) [11]."

An example of the constructive process is given in Figure 11. We have to create a score for 4 instruments. The first step must choose a constituent *musical action* for the staff 1. After the comparison of different *gestalt patterns,* in step 1 and 2, in the third step (marked by the symbol "*") is chosen the *function to music* (*sound pattern*) of an hypothetical piano part which helps to show how *sound patterns* are involved in musical organization. The vibrating strings of the piano are interpreted by the algorithm as a decay. This can be used by the first line in order to adjust its behavior.

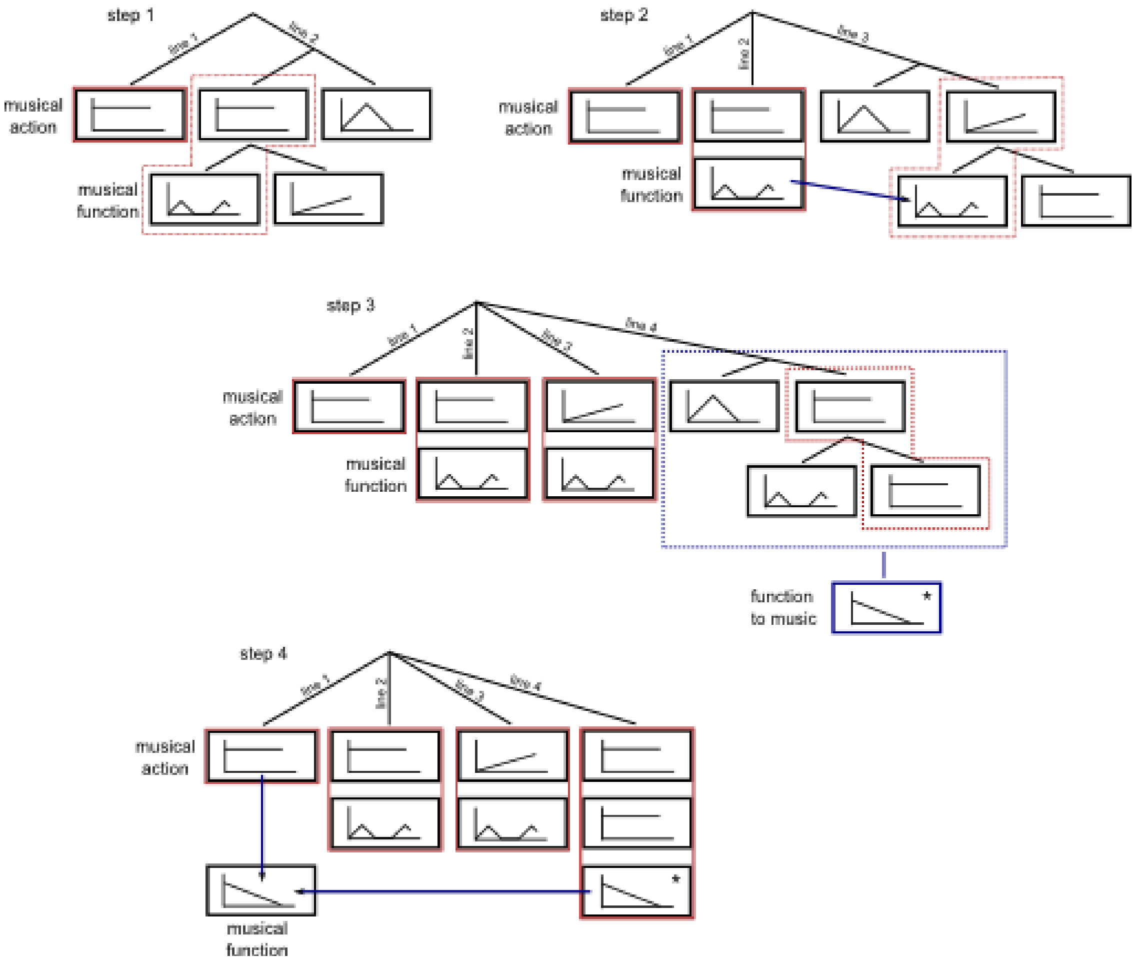


Figure 11: Structural algorithm example.

If we add a similar melodic model to the score, as in Figure 12,

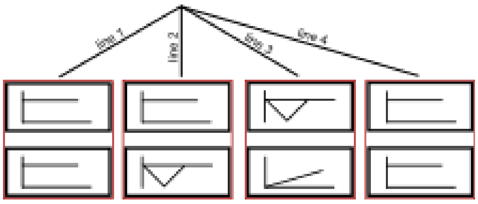


Figure 12

a possible outcome could be:

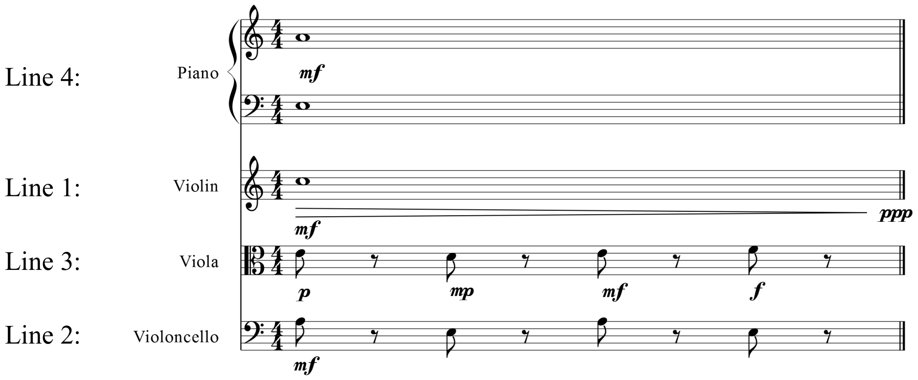


Figure 13: Score outcome example by means of structural model.

At the same time the use of subtrees with different sizes gives us a powerful mean for musical structuring. As we said in chapter 2 (Figure 4), an *action pattern* (and *sound* and *score patterns* as well) can be identified by the description of its generative *gestalt matrix* and the associated scalar.

This means two things: Firstly, the melody of a musical fragment can be composed as a sum of different patterns. A semibreve bass line can be associated to a melodic line formed of two consequent sessions. These can have different patterns both related to the constituent one (the bass line). Secondly, each pattern can be associated to a certain musical action's length. For example, if an eight note is represented by an *action pattern* of size two (i=2), the proportioned size of a whole note is sixteen. A pattern can repeat itself 3 times, making the system more flexible.

By use of other probability theories, such as *Markov's Chains,* it is possible to connect each line development to the previous pattern model in order to give continuity to the entire system.

6 Likelihood-DOP probability model

The use of Likelihood-DOP probability model [10] seems to enrich the system behavior to give unity to the score development and to create appropriate relationships between patterns, permitting models, or part of them, to be suggested during the performance.

The probability of a subtree *t, P(t),* is computed as the number of occurrences of *t*, $|t|$, divided by the total number of occurrences of treebank-subtrees that have the same root label as *t*. Let *r(t)* return the root label of *t*. Then we may write:

$$P(t) = \frac{|t|}{\Sigma_{t':\, r(t')=r(t)} |t'|} \qquad (1.11)$$

Set t_n the *n*th instrument treebank-subtree, time by time, thanks to the enrichment of the score treebank, the probability of an incoming musical fragment $t_1 \circ \ldots \circ t_n$ is given by [10]:

$$P(t_1 \circ \ldots \circ t_n) = \Pi_i \; P(t_i) \qquad (1.12)$$

The variance and originality of the score is given by the balance of new pattern relations and Likelihood-DOP probability feedback.

Finally but nonetheless importantly, because of the tree-family typology it belongs to, every score fragment and each involved pattern can be reduced or stretched, thus creating micro and macro variation similarities during the construction.

Defined so the creation of one or more *score actions,* the interaction between the various instrumental staffs permits the modification/reconstruction of sub-patterns (*sound* and *action* ones), and the association of different *score patterns* creates relationships between sub and super-tree levels.

The research of an emotional application can be pursued by use of Likelikihood-DOP probability model.

Emotions derive from big or small changes in the musical structure. The more the following sessions are far from the most predictable, the more the listener will be brought to feel some emotion.

After a *crescendo score pattern, it* is probably more common to think that the next incoming fragment will have a *score pattern* similar to the precedent or a flat one, represented in Figure 5. But if a succession ending with a dominant chord is presented, it is possible to lead the algorithm to choose the less probable continuation of the pattern exposed. Thus, *ff* can be transformed in *p* or *pp*. At the same way a rhythm on long-value notes can be opposed to a fast execution and the number of instruments can be reduced from *tutti* to one or two.

In Figure 14 is shown the simplified application of the general model proposed.

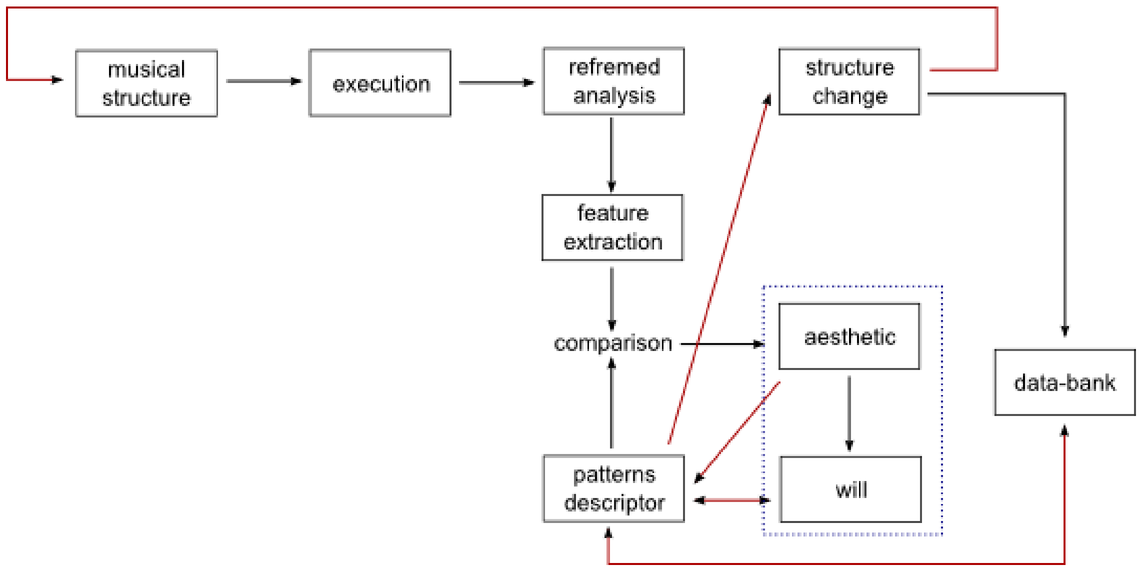


Figure 14: Model application.

In Figure 14, from the assumption that a musical fragment is already given, the *musical structure* is the score fragment which the instrument is going to perform. The *execution* block gives to the algorithm (whenever an efficient model is implemented in the algorithm) the property of applying musical deviations on the instruments playing to make the execution as real as possible. The two steps encircled by the blue dotted line have to be multiplied by the total number of musical lines used and, to strengthen the system response, the aesthetic model has to be applied to the final outcome. The *pattern descriptor* is the theoretical model reported in Figure 10.

6 Conclusions

Musical actions can easily be applied to the score during the execution by a synchronized substitution of the matrix. The fast changing of musical actions, in real time application (whenever is possible a real-time execution), appears useful if related to gesture recognition algorithms.

It is impressing the theoretical richness the outcome could give, by means of the idea that the same sound can be chosen or described by different paths.

It seems that the model can be applied to most of the musical properties describing the piece. For example, a harmonic construction, associated with an impulsive pattern, can be translated in a harmonic fragment where the tonality is left, to be back at the end of the fragment, if no *alerts* are compiled.

References

[1] Gardner, M. (1970). The Fantastic Combinations of John Conway's New Solitaire Game 'Life'. *Scientific American.* **223: 120-123**.

[2] Ganguly, N., Sikdar, B. K., Deutsch, A., Cantight, G., Chaudhuri, P. P. (2003). A Survey of Cellular Automata. *Technical Report*, Centre for High Performance Computing, Dresden University of Technology, 2.

[3] Smith, A. (1976). Introduction to and Survey of Polyautomata Theory. *Automata, Languages, Development*, North Holland Publishing Co., **405-422**.

[4] Banks, E. R. (1971) Information Processing and Transmission in Celular Automata. *PhD thesis*, M.I.T.

[5] Codd. E. F., (1968). *Cellular Automata* (First Ed.), Academic Press Inc.

[6] Flury, J., Bisig, D. (2006). Celerina A Generative Music System Using Aesthetical Reduction Applied to Simple Cellular Automata. *Proceedings of the 19th International FLAIRS Conference*. Melbourne, USA.

[7] Wolfram, S. (1983). Statistical Mechanics of Cellular Automata. *Reviews of Modern Phy sics,* **55: 601**.

[8] Smith, J. O. (1992). Physical Modeling using Digital Waveguides. *Computer Music Journal*, **16: 4**, 1.

[9] Smith, J. O. (1996). Physical Modeling Synthesis Update. *Computer Music Journal*, 20, no. **2: 44-56**, 2.

[10] Bod, R. (2002). A Unified Model of Structural Organization in Language and Music*, Journal of Artificial Intelligence Research*, **17: 289-308**.

[11] Manning, C., Schütze, H. (1999). Foundations of Statistical Natural Language Processing. Cambridge, *The MIT Press*

Beyond Markov Chains, Towards Adaptive Memristor Network-based Music Generation

Ella Gale[*], Oliver Matthews,
Ben de Lacy Costello and Andrew Adamatzky
The University of the West of England

March 5, 2013

Abstract

We undertook a study of the use of a memristor network for music generation, making use of the memristor's memory to go beyond the Markov hypothesis. Seed transition matrices are created and populated using memristor equations, and which are shown to generate musical melodies and change in style over time as a result of feedback into the transition matrix. The spiking properties of simple memristor networks are demonstrated and discussed with reference to applications of music making. The limitations of simulating composing memristor networks in von Neumann hardware is discussed and a hardware solution based on physical memristor properties is presented.

1 Introduction

The human universals [1] are traits found in all human cultures since the Upper Paleolithic and are unique to humanity; music is on this list of around 370 concepts and behaviours including examples like: dance, hope, language, fire, fear of death, cooking, prohibition of murder, hairstyles and other behaviours both similarly dramatic and banal. Music's role in human culture is related to sexual attraction, social cohesion, relaxation and communication (see [2] for a recent review from the anthropological context). It is believed that, like some other human universals, music may be a product of the structure of the mind [1], and thus a by-product of human evolution. However, the popular

---

[*] Corresponding author

idea that music is a universal language or pre-language has been resoundly disproved, as far back as 1940 [3], by cross-cultural studies that showed that the emotional resonance of music is a culturally learned response.

The combination of two human universals, anthropomorphisation and tools, would suggest that the best tool would be human-like and thus it's not surprising that, after the invention of computers, artificial intelligence, A.I., (namely the desire to create an intelligent machine) would be an area of active research. A.I. has had some successes such as learning-classifying systems and neural networks, however the creation of creative A.I.s has had fewer successes, and the harder problem of creating a self-aware and conscious machine intelligence has suffered from even less progress.

Music generation is a good problem to tackle if one is interested in making a creative A.I., furthermore, if music does arise as a result of brain structure, then it might be fruitful to approach the problems of neuromorphic engineering (that of making brain-like computers) by creating a composing brain utilising a human-selection process on the output music: it's easier to recognise a melody than brain-like activity in a neural network. This approach will have the added drawback (or perhaps benefit) of adding a cultural bias to the music.

The study of creativity is a large and multidisciplinary area, with competing definitions and a lack of consensus, however, a recent attempt at formalising creativity and describing the actions of creative A.I. agents requires two modes of learning: A. an adaptive predictor or model of the growing data history and B. a reinforcement learner [4]. The agent learns about the world around it, compresses and stores that data and as such makes predictions about the future, this model encodes the known structure of a certain style of music, thus making the output similar to music in the same style, but by using reinforcement learning tuned to novelty, the new music is different enough to be interesting.

Markov chains have been well exploited in music generation [5]. The Markov Hypothesis states that a future state of a sequence depends only on the last state. Usually a matrix of note transitions is seeded with a corpus of music of a particular style and music is generated via a random walk. While often effective, it has a number of drawbacks; the biggest is that music has an underlying structure and requires long-term order which contravenes the Markov Hypothesis [6].

Unconventional computing is a branch of computing that aims to go beyond the von Neumann models of computation and includes, but is not limited to, methods like chemical computation, biological computation, cellular automata, quantum computing, neuronal computing [7]. Various unconventional computing approaches have been applied to music generation, such as cellular automata music generation, sonifying *Physarum polycephalum* and sound synthesis using a neuronal network (wetware). Using memristors is considered a unconventional computing approach due to their novel communication interactions [8] and similarity to the neurons [9, 10].

Memristors are the recently discovered [11] 4^{th} fundamental circuit element [12]. The memristor changes its resistance as a function of the amount of charge that has passed through it (which is also proportional to voltage). Unlike the other three fundamental circuit elements (the resistor, capacitor and inductor) the memristor is non-linear and possesses a memory. Memristors have also been compared to neurons in the brain due to their spiking response to changes in input.

For the auto-generation of music, we are interested in four properties of memristors: their non-linearity, their time-dependence, their memory and their spiking response. As a network of memristors would necessarily possess a memory that goes beyond the previous state [1], music generation using a memristor networks offers us a route to go beyond Markov chaining.

Memristors have been used as synapse analogues in STDP (Spiking Time Dependent Plasticity) neural networks [13]. Here, we plan to use memristors as the connections between a graph of musical notes, where the memristors can modify their connection weight non-linearly with the number of times one musical note follows on from another in a piece of seed music. This will create a weigthed graph, which can be built in the lab by connecting memristors. In this paper, we shall simulate such a graph by simulating the memristor connections, as in [14].

A network of memristors can spike, and these spikes are believed to be deterministic and related to the change in voltage across a memristor. These voltage changes and spikes can propagate across a network through time in a complex manner (as the voltage change from one spike will cause a voltage change in memristors further along the network, causing further spikes and so on). Thus, the spike interactions can be used to ‘play’ music by choosing which notes follow on from one another (in a Markov chain approach).

There are two timescales that interact in a memristor network and which can give rise to altering tempo of played notes. The first is the relaxation time of a memristor after its spike (this is related to the memristors memory). The second is related to the length of the wires between the memristor and the time taken for a signal to propagate from one spiking memristor to the rest of the network.

These three aspects: the seeded network, the spikes and the time depen-

---

[1] technically the memristor’s memory is dependent on its entire history from $-\infty$ to now, in practice it is possible to ‘zero’ a memristor’s memory

dence can interact to give complex behaviour. However, each spike will alter the structure of the network, allowing the system to change over time, leading to developing new patterns in its musical style to 'evolve' (or possibly allowing it to get stuck in a stable attractor).

In this paper, we will discuss the methodology and challenges for building such a network, by simulating a demo network, demonstrating the spikes responses in a simple real memristor network, simulating a simplified version of the spiking seeded network and finally discussing whether such a spiking seeded network can be fully simulated in a computationally tractable way using standard von Neumann architecture.

2 Building the Memristor Network

2.1 Setting up the graph

For this work we will consider a musical range of only two octaves, stretching from C4 to B♭5, for a total of 24 notes. As any note could potentially follow on from any other, the graph of all possible links would be a reflexive directed k-graph of 24 nodes and 576 vertices. We show, as an example, a fully connected directed k-graph for 12 notes (an octave) in figure 1. For comparison, a network for a full standard piano would require 88 nodes, requiring 7744 memristors to model, as shown in figure **??**. In the real network, a transition (e.g. A5→B♭5) would be recorded by an ammeter in series with the memristor).

Similarly, the timing of the notes was also constructed by a network. In the actual device, we would expect the memristor spikes to provide the tempo information, for out simplified model a second (much simpler) network built for the tempo analogously as for the notes in the melody. The tempo was broken down into 9 components: semiquaver, quaver, crochet, minim, their dotted versions and the breve.

The memristor network would be held at a constant voltage and as the memristor spikes, these spikes then propagate around the network, each spiking memristor is transiently the source of the ΔV perturbation and each other memristor is the drain. In our simulation, this will be modelled by moving the source to the node associated with the previously played note at each step.

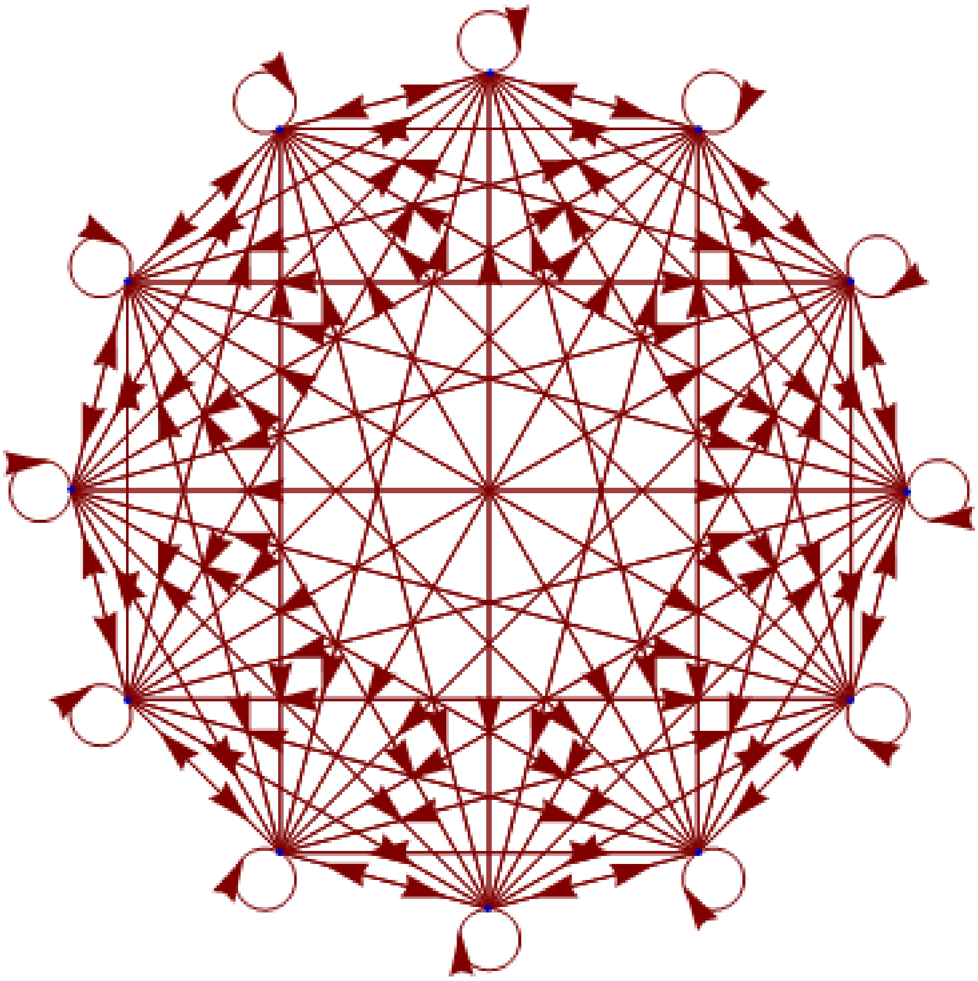


Figure 1: A self-linked, complete directed k-graph. Note that the forwards and backwards connections are drawn to overlap here.

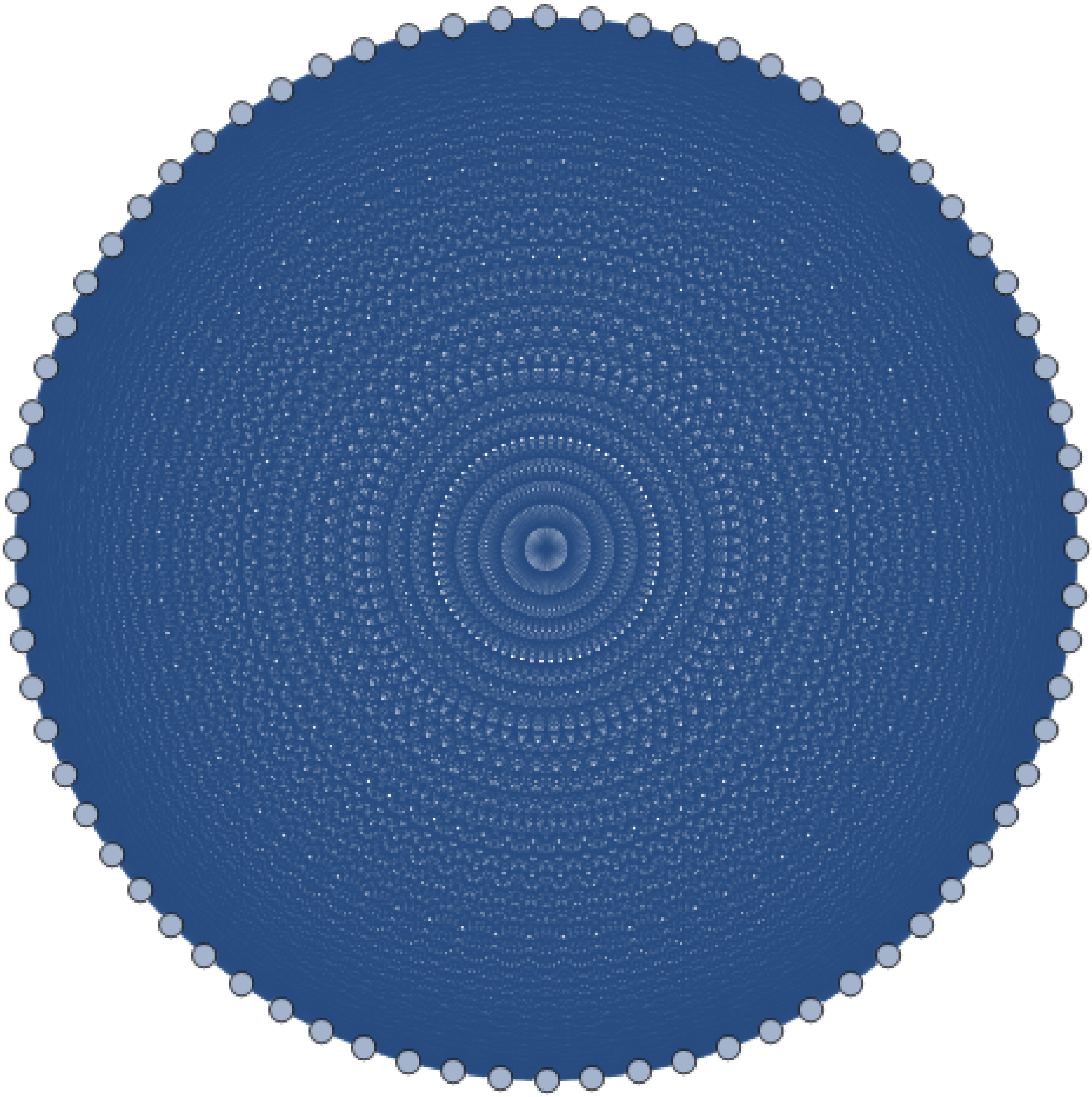


Figure 2: Complete k-graph connected for the 6 octaves of a piano. The self-connections have not been drawn for clarity.

2.2 Memristive Connections

The ‘probability’ of a transition between two notes occurring will be related to the connection weight of that vertex. In the memristor network, the connection weight is the conductivity of the memristor. Figure **??** shows the conductivity of am memristive connection increasing under a constant voltage (with a linear conductivity profile shown for comparison). As each transition is either heard or created, the memristor conductance moves up this curve. As music is a directed graph (it matters whether we go from C→A or A→C) there will be a second memristor, wired up the other way round, which goes down the curve. This property means that the reverse transition is less likely if the melody has just performed the transition.

Memristance is defined [12] as:

$$V = M(t)I \ ,$$

we will use the Mem-Con model of memristance [15]

$$M(t) = M_e(t) + R_{\mathrm{con}(t)}$$

which has the advantage of having been fit to the devices in our lab, so we can later use measured experimental values as in [16]. In this paper, we use reduced units, i.e. the conductance is measured in terms of device properties such as the size and resistivity of the material. The conductivity of a memristor, $G(t)$, is given by

$$G(t) = \frac{1}{M(t)} \ ,$$

and as the connection weight in the graph is simply the conductivity, it is also represented by $G(t)$.

2.3 Seeded graphs

To seed the melody network, we converted several different pieces of musical melody to a list of notes in the key of C Major, transposing to the key of C and mapping them to the two octave range we have available. To seed the tempo network, the tempo was converted to reduced units where a time of ‘1’ is equal to 1 crochet, this allowed us to normalise for beats-per-minute variations between the seed music. This approach divorces the tempo from the notes, which we felt was accurate as melody rarely correlates the speed of the note to its pitch (however as many a base singer will say, the backing

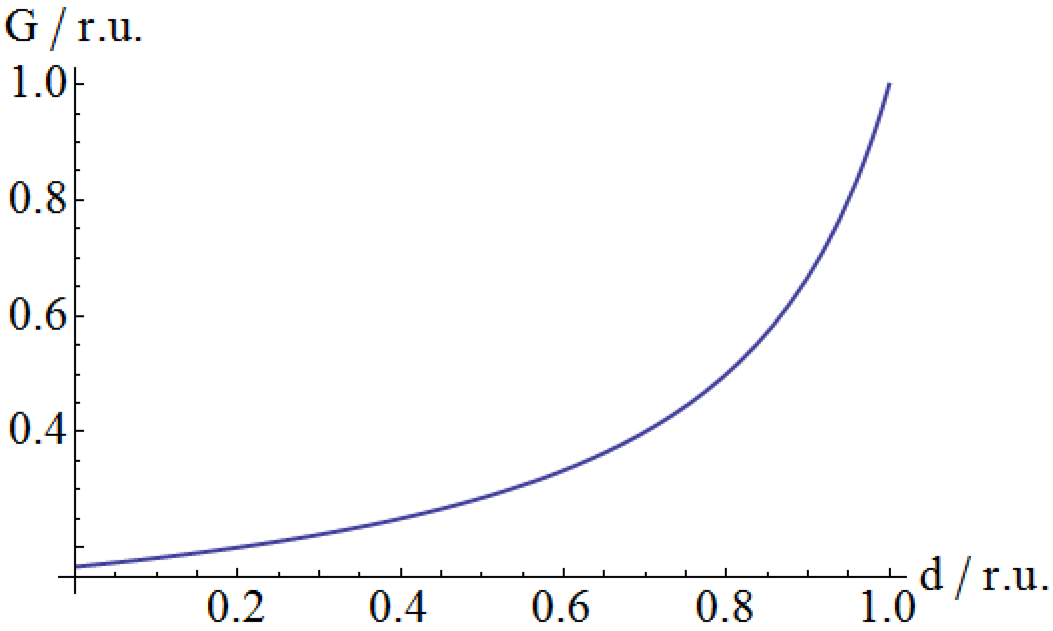


Figure 3: Conductance profile for a continuously charged memristor. This is also how the connection weight for a memristor-based connection.

baseline of harmonising choral pieces is usually the beat and thus includes less variations in the tempo).

From a frequency analysis of the number of times a transition happened, a transition matrix was populated with the expected conductance values as based on the memristor conductivity curve in figure **??**. Different musical seeds created networks with different structures. We chose to investigate three distinct styles of musical melody, namely jazz standards, rock'n'roll as exemplified by Elvis (his faster tracks) and light opera as exemplified by Gilbert and Sullivan. The three Jazz standards were: 'How high the moon','Ain't that a kick in the head', 'I've got the world on a string'. The Elvis tunes were: 'All shook up', 'Burning love' and 'Jailhouse rock'. The chosen Gilbert and Sullivan classics were three solos taken from Pirates of Penzance: 'Modern Major General', 'When a felon' and 'Better far to live and die'. Specifically, the primary vocal line was taken for each as the melody.

For example fig 4 shows example graphs for the melody lines for the three jazz standards, specifically. The graphs tend to be sparse as a single melody is repetitive and does not cover a huge note-space. The largest connection weights tend to be on or close to the diagonal due to the fact that repetition of a note is common when singing a phrase, and because the further from the diagonal the larger the jump between the notes and the human voice has

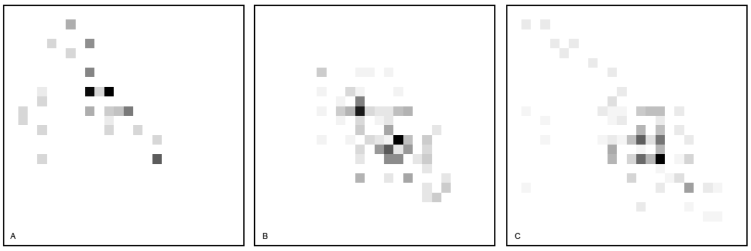


Figure 4: Example connection matrices for memristor networks seeded with the melody line of jazz standards: A, How high the moon, B, One for my baby, C Ain't that a kick to the head. The darker the colour, the more often that transition is heard.

difficulty with larger jumps.

3 Music Generation

Having seeded our network, we will now discuss how to use it to create music. We shall start by examining how memristor networks act, and then consider the approximations that must be made to model them in simulation.

3.1 Memristor Networks

Fig 5 shows an example of memristor spikes. There are a recovery time when the system relaxes to it's long term value. This takes around [20]s and may be tunable by varying device parameters. Generally, the larger ΔV the larger the spike.

Intriguingly, these spikes give rise to complex behaviour. Consider the circuit in figure [**?**] A d.c. voltage source is applied from a Keithley 2400 sourcemeter (drawn as a battery, as is standard in such circuits), for a single memristor this would apply a sharp step function from 0V to the set voltage at the first step and then hold it there. Figure 5 shows the response of a memristor to such a voltage. When there are multiple memristors in a circuit, the spike from the voltage application isn't there, instead you get complex behavior like that seen in figure 6, this is response of ammeter A_{T}. We suspect that this is due to the sudden voltage step occurring at slightly different times across the 3 memristors. Each current spike causes a change in resistance across the memristor it reaches, which causes a change in voltage ΔV which then causes another current spike, this can bounce around the network indefinitely if provided with an energy source (namely

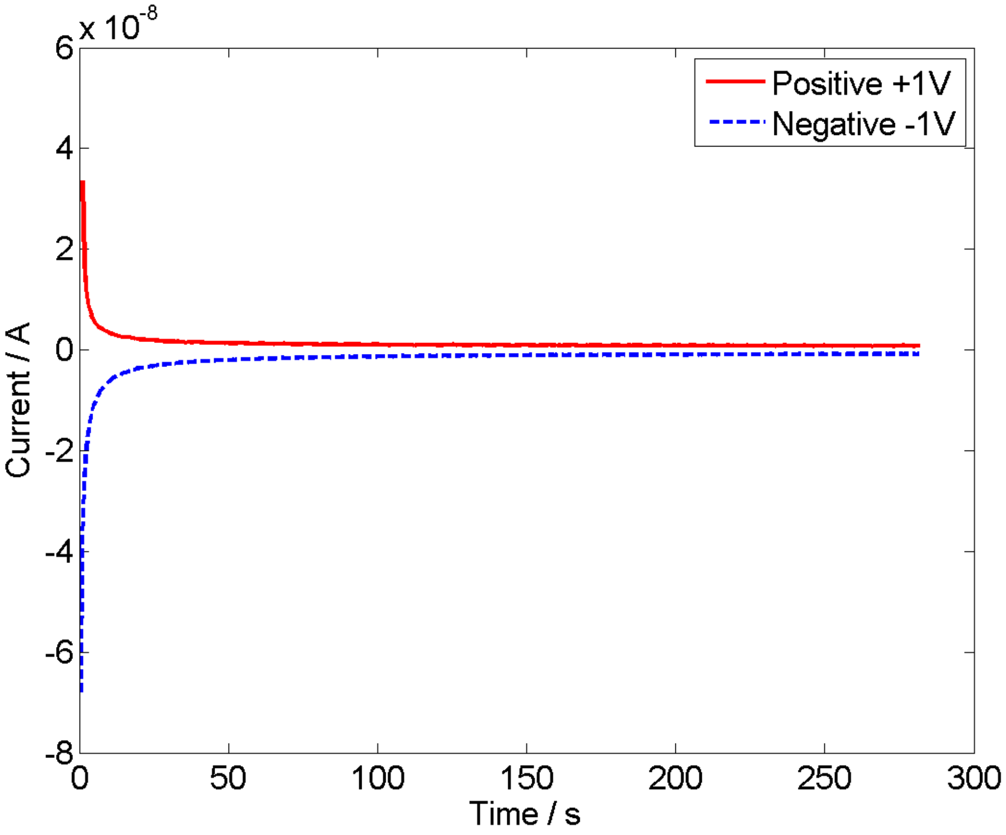


Figure 5: Example positive and negative spikes.

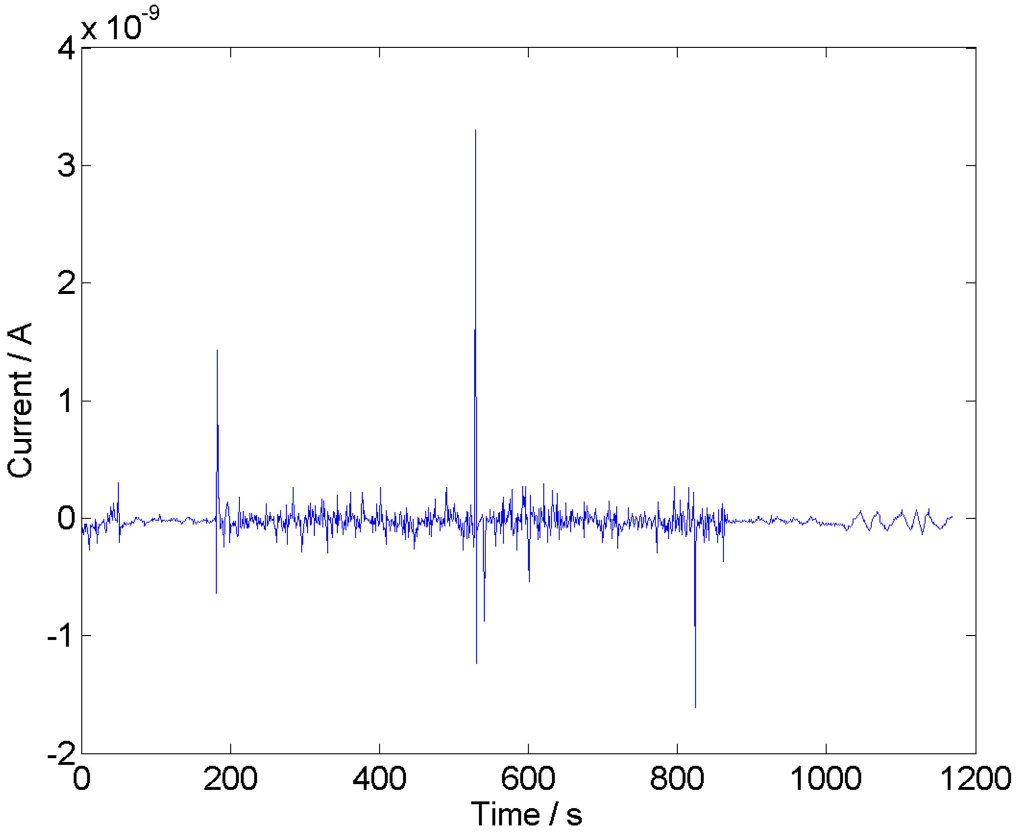


Figure 6: An example of spike patterns through a very simple memristor network

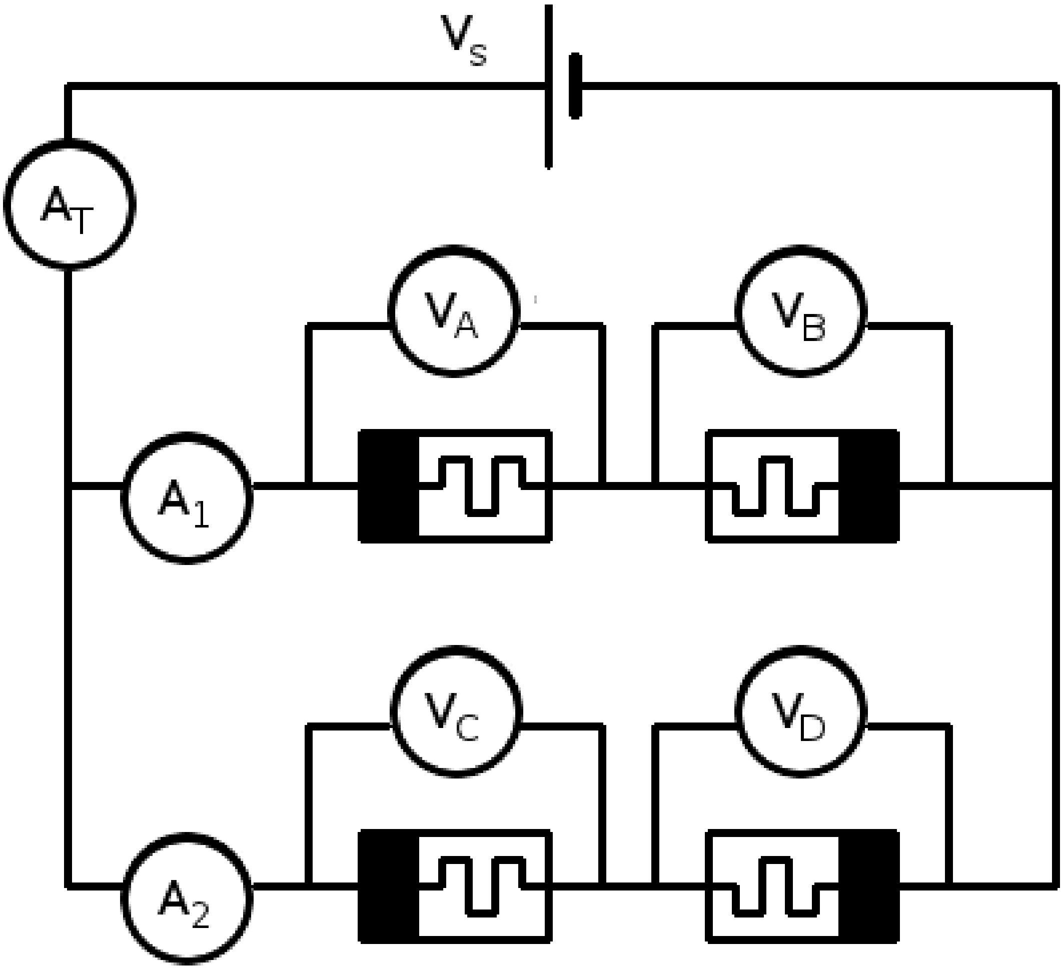


Figure 7: A scheme for two note transitions as an example.

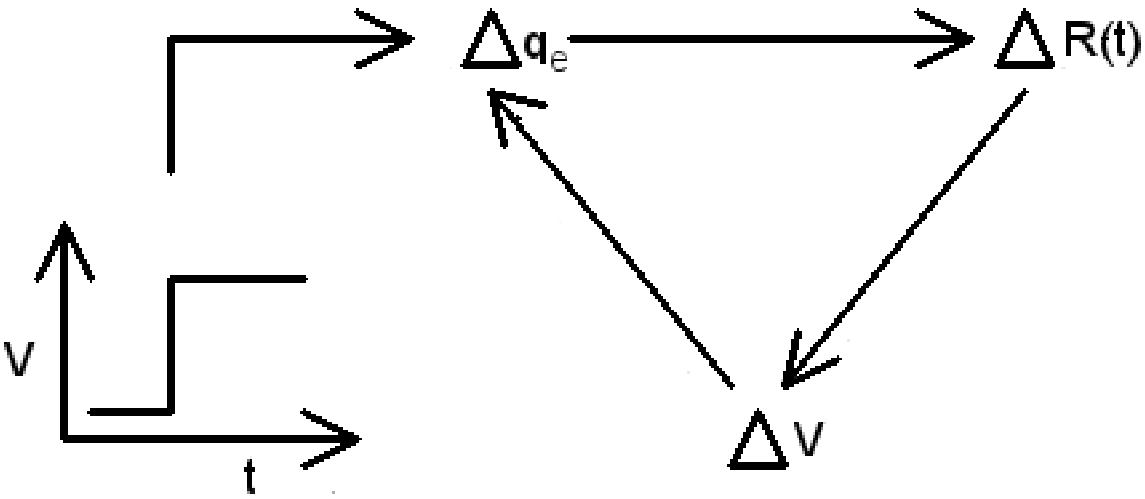


Figure 8: The process of spike creation and continuation.

the applied constant voltage). There are two routes by which the memristors can interact, the first is the creation and movement of current spikes, Δq_{e^-}, as in extra charge, q, is drawn from the source, and this alters the resistance. The second is a change in resistance, δR which causes a change in voltage (as $\frac{1}{\frac{1}{V_A+V_B}+\frac{1}{V_C}}$, which itself causes a current spike. Figure [?] summarises this interrelation.

3.2 Time dependence

The time dependence due to the delay in signal (current) propagation in a memristor network has been discussed. The other relevent time is the relaxation time, τ_r. When a memristor representing a transition from $X \to Y$, where $X, Y \in \{CD\flat DE\flat EFG\flat GA\flat AB\flat B\}$, spikes and alters its resistance, the reverse transition, that of $Y \to X$ is slightly inhibited because both memristors between nodes X and Y have been altered. The lifetime τ_r defines how long the memristors take to equilibrate. Before that point, the spikes are smaller if in the same direction and larger if in the opposite way.

These interactions in timing will also cause the spikes to occur at a non-regular rhythm, avoiding boring musical tempo. However, the interactions that give rise to the oscillations lead to concept of a beat.

4 Modelling a spiking memristor network for music creation

If we have a seeded memristor network and turn on a constant voltage, we should create spiking network. If each spike is taken as making a transition from one note to another, then the network's activity will generate music. Also, as the network generates music, it is also learning, so the network will respond to the music it creates and alter the connectivity of the k-graph and thus the music created by the network.

4.1 Problems with attempting to model a memristor network

There are two main problems with modelling such a network. The first is the problem of modelling transient δV. This is relatively easy to solve. In the real network we would set the wired network up and record what it produces. In the simulation, we don't need a background voltage to power the simulation and can thus set the current note as the source (and itself and all others as a possible drain).

The second problem is more intractable. We need to know when and if a memristor will spike. It is not known if the spikes are probabilistic or deterministic in nature; this is a current area of investigation. To model to the system as deterministic, we would need to descretise time very finely and model the state of everything in the network at once. If the system is chaotic or near the edge of chaos (a very real possibility) any approximations or course-graining of the system will result in an extremely inaccurate simulation. Furthermore, it is not currently known what causes the spikes, we suspect that those measured in fig 6 are the addition of spikes from the individual memristors. But we don't know what causes these individual memristors to spike or not spike. Also, the network as drawn in part one of this paper, is not a standard electronic circuit that can be entirely resolved into series and parallel relationships, making the network as a whole non-simple and difficult to model.

Finally, for a one octave memristor network we require 144 memristors, to do an entire piano's range we would need 7,744 (plus 81 for the tempo). We can not necessarily make the approximation of considering one memristors against a 'mean-field' background of the other memristors, due to the almost instantaneous [2] ΔV, thus even with our simplified version we have a 576-body

---

[2] As energy can not be created or destroyed, a change in voltage should change the voltage drop across the rest of the network instantaneously. Whether this change is actually

problem to attempt to calculate. We suggest that solving such a problem with standard von Neumann computer architecture will be computationally intractable (although it is not theoretically impossible).

Our obvious solution to these issues is to build a non-von Neumann computer architecture, i.e. to actually build a network of 576 memristors. However, before undertaking such an endeavour, it's worth doing a preliminary, highly simplified simulation to check that a seeded memristor network would be of use to generating music and to see if the non-linearity of the memristor model by itself does not offer interesting aspects to procedural music composition.

4.2 Using a simplified memristor network model to perform non-Markovian music generation on a pre-seeded network

As described in the previous section, the 'roving' ΔV will be modelled as V against a background of $0V$, in that each note will be set to the voltage source in turn. This is a gross simplification of the laws of physics but will serve to course grain the effects of a network. The drain will be connected to the drain of all other nodes (including the drain of that node, which allows for a self→self transition.

There is a function $p(t)$ which controls whether a memristor will spike or not. We suspect that the memristor network is deterministic and chaotic in form, but have no current knowledge of this function. Therefore, we shall take the simplification of assuming that the pseudo-random chaos can be coarsely representated as the pseudorandom values from a Gaussian random number generator. Thus, we will talk about the probabilities of a transition between X and Y occurring, $p_{(X \to Y)}$ as being a product of the connection weight and our unknown function $p(t)$, which is itself set as a pseudorandom number, $p(x)$. Thus,

$$p_{(X \to Y)} = G_{(X \to Y)} p(x) \ .$$

And the next note, $n+1$, is determined by the maximum of this product over the set of all a possibilities, i.e.:

$$p_{(n \to +n)} = \mathrm{Max}[\{p_{(X \to Y)} : X, Y \in \{C4 : B\flat 5\}\{p(x)_T\}] \ ,$$

where T is the set of all 576 possible transitions.

After a given connection has fired, we slightly increase it's state along the memristive curve (to reflect the current that flowed through it as part of the

---

instantaneous or proceeds at the speed of light is a question for relativity physicists

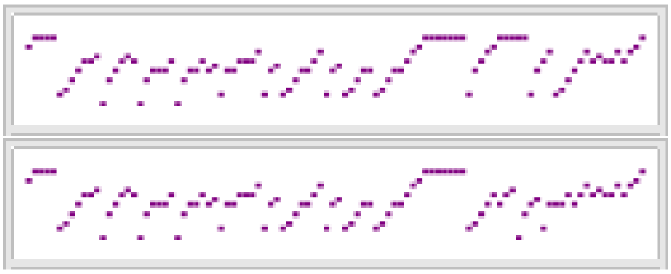


Figure 9: Generated music based on the 'How high the moon' connectivity matrix. Top is without feedback into the connectivity matrix, bottom included feedback, both have the same pseudorandom number input. The connectivity matrix is altered and the alteration is slow.

spike) and use this new state for calculating future transitions. To model the relaxation time, the memristor that has spiked on step n is artificially moved down the memristor curve to a quarter of it's value for step $n+1$ and half for step $n+2$, this substantially reduces the likeliness of it firing again until step $n+3$ where it is set to its new (increased) weight. To model the reverse note connections (and prevent the over-occurance of the odd musical structure of $X \rightarrow Y \rightarrow X \rightarrow Y \rightarrow X ... \infty, X, Y \in \{C4 : B\flat 5\}$, the reverse transitions are decremented rather than incremented at step n and similarly reduced to a quarter and half their new value on steps $n+1$ and $n+2$. The music is started on note $C4$ and the first note of the tune taken from the first transition from that note. 100 notes were generated for each tune.

Figure 9 shows an example of generated music from the connectivity matrix seeded with 'How high the moon'. Despite the simplicity, it sounds like music rather than random notes. The top subfigure shows music output from a static matrix, the bottom shows the effect of allowing the transition matrix to be seeded by the musics it's generating, and thus can only be seen at the end. This is what we want, as we don't want the music generator to change too quickly.

Figure 10 shows further examples of the plasticity of the transition matrix. This figure clearly shows the strong effect of the transition matrix on the music generated, but we can see that over time and repeated generated notes, the melody is changing.

4.3 Results from our modelled network

The combined connection matrices for the note connections and note lengths are shown in figures 11 and 12 respectively. Simple examination of these

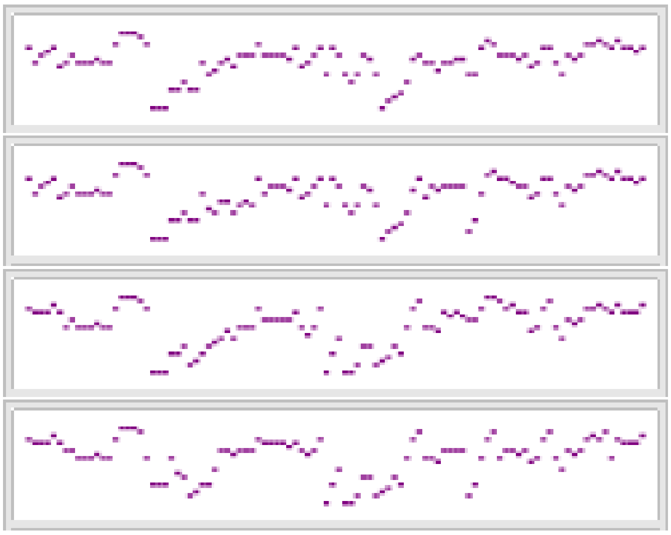


Figure 10: The effect of feedback into the transition matrix. Here the matrix used was the combination which was seeded by all three jazz standards. In order, they are melodies generated by the original triple matrix, and those generated after 1,000,10,000 and 100,000 notes generated respectively.

matrices can tell us a few things about the differences between these musical styles. Looking at the tempo changes in figure 12 we see that the jazz standards make the most use of different length notes with the crochet and quaver being the most popular. Both the rock'n'roll and the light classical are 'faster' as they use quavers overwhelmingly (note that this doens't apply to all music in this genre, we chose rather fast classical and Elvis' more dancable tunes). Oddly, the light classics had less variation than rock'n'roll in the timing.

As the graphs shown in figure 11 are less easy to understand at a glance. Elvis's rock'n'roll tracks tend to focus on either the lower notes or the higher ones. Both the jazz and the light opera avoids the higher notes. We can look at the reducibility of the connection matrices, which is a measure of the minimum number of connections to represent this music (i.e. pruning the unused connections). For light opera melodies we only need 16, 19 for the rock'n'roll and 20 for the jazz standards (perhaps reflecting that the jazz standards were not the product of a single composer or composition team). The matrices are not symmetrical, but are not far off it and the symmetry can be measured by taking the difference between the transpose and the original matrix. All three styles have a similar symmetry, with the light opera being 83% symmetrical, the jazz standards are 83.3% and the rock'n'roll is the least symmetrical at 85.4%.

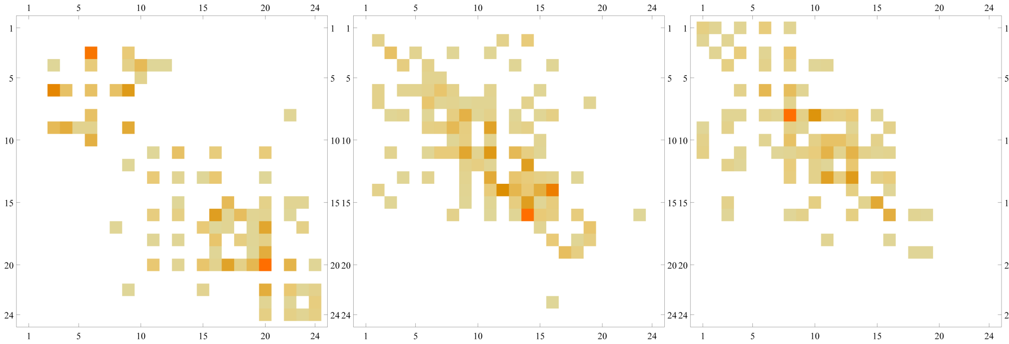


Figure 11: The connection matrices as seeded from three pieces of music in each genre: A. is Jazz standards, B is the Elvis rock'n'roll and C is the Gilbert and Sullivan light opera.

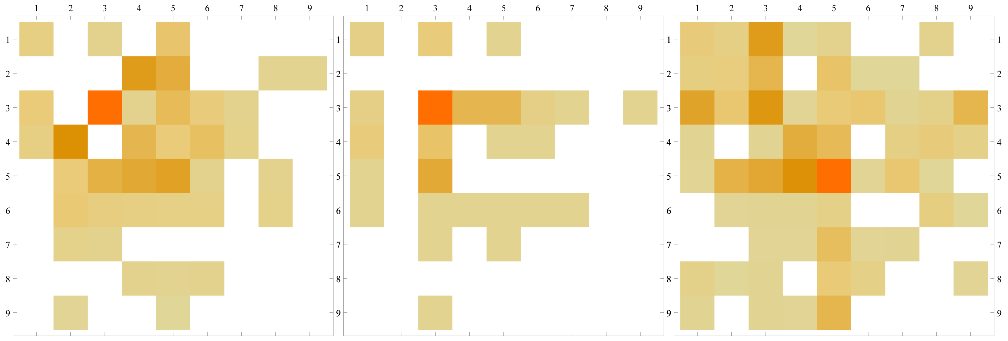


Figure 12: The tempo connection matrices for seeded with three melodies from three separate genres: A is Jazz standards, B is the rock'n'roll and C is light opera.

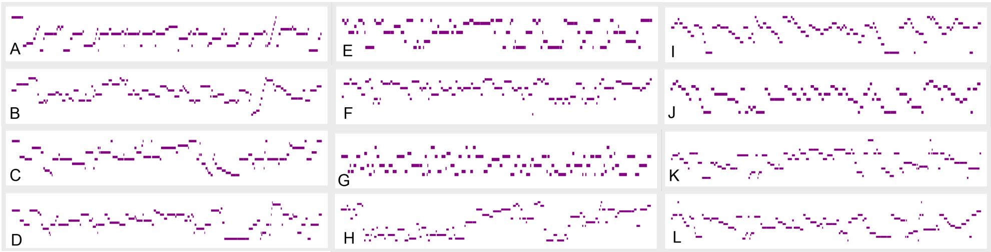


Figure 13: The composed music, output as a result of different seeded connection matrices. A. How high the moon, B. Ain't that a kick in the head, C. The world on a string, D. seeded with all three jazz standards, E, All shook up, F. Burning love, G. Jailhouse rock, H seeded with all three Elvis tunes, I. Modern Major General, J. When a Felon, K. Better far to live and die, L connection matrix seeded with all three gilbert and Sullivan melodies

Figure 13 shows the output music from memristor networks as seeded by single songs (on the top three rows) and connection matrices seeded by all three input songs. Each of these graphs was generated with the same pseudo-random generated numbers.

Note, that the system we have set up has been only used to generated the melody, however the system as set-up can allow multiple simultaneous connections, i.e. chords. In a real memristor network, we would expect multiple connections to spike at the same time, creating chords.

The separation of tempo from note, allows us to compose music (using both) and change the performance of the piece using the tempo matrix.

5 Time variance of this system

As explained, the time dependence and spiking behaviour will feedback into the structure of the network and thus alter the system over time. This should make a robust and interesting music generating system, but may produce static solutions such as a dead network (no spikes), epilectic network (many spikes in utter synchrony), resting network (spikes in a repetitive, boring, oscillation). As these solutions have not yet been observed in memristor lab tests, we are optimistic about avoiding such solutions. Nonetheless, all three of these stable state solutions can be reset by wiping and reseeding the network.

6 Conclusions

We have demonstrated a novel approach to music generation networks that relies on non von Neumann hardware and is computationally intractable to solve. We have shown with our simplifications that it is worth pursuing. We have identified properties of memristors which are useful for such a network and tested simple 3 memristor proof-of-concept prototypes.

The problem of building a memristor-based music composer is an interesting one as it involves many of the same challenges as building a memristor-based brain. Perhaps the creation of recognisable music is a good test of the creation of a working brain. Furthermore, the problem of creating music is far more tractable than that of creating a brain, as every human can recognise a good end solution ('Does it sound like music or noise?), whereas no one can really recognise a working intelligence from looking at the spikes in a neuronal network.

The creation of a 7,744 memristor network is a little beyond the current state of the art. In our lab, memristors are synthesised by the hundred and have to be wired together by hand. However, such a network is not that far off at places like HP where they have successfully synthesied many thousands of memristors in a neural memristor chip. It is currently outside of the price range of a music composer, but we anticipate the price to come down in the future.

References

[1] D.E. Brown. Human universals. *Temple University Press*, 1991.

[2] Miriam Cihodariu. A rough guide to musical anthropology. *Journal of Comparative Research in Anthropology and Sociology*, 2:183–195, 2011.

[3] Robert Morey. Upset in emotions. *Journal of Social Psychology*, 12:355–356, 1940.

[4] J. Schmidhuber. Formal theory of creativity, fun and intrinsic motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, 2:203–247, 2010.

[5] F.P. Brooks Jr., P.G. Neumann A.L. Hopkins Jr., and W.V. Wright. Mit press. *An Experiment in Musical Composition*, pages 23–40, 1992.

[6] P. Roy F. Pachet and G.Barbieri. Finite-length markov processes with constraints. *Proceedings of the Twenty-second International Joint Conference on Artificial Intelligence*, pages 635–641.

[7] A. Adamatzky and C. Teuscher. From utopian to genuine unconventional computers. *Luniver Press.*

[8] Ella M Gale, Benjamin de Lacy Costello, and Andy Adamatzky. Observation and characterization of memristor current spikes and their application to neuromorphic computation. In *2012 International Conference on Numerical Analysis and Applied Mathematics (ICNAAM 2012)*, Kos, Greece, Sept 2012.

[9] Leon Chua, Valery Sbitnev, and Hyongsuk Kim. Hodgkin-huxley axon is made of memristors. *International Journal of Bifurcation and Chaos*, 22:1230011 (48pp), 2012.

[10] Leon Chua, Valery Sbitnev, and Hyongsuk Kim. Neurons are poised near the edge of chaos. *International Journal of Bifurcation and Chaos*, 11:1250098 (49pp), 2012.

[11] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams. The missing memristor found. *Nature*, 453:80–83, 2008.

[12] L. O. Chua. Memristor - the missing circuit element. *IEEE Trans. Circuit Theory*, 18:507–519, 1971.

[13] D. Howard, E. Gale, L. Bull, B. De Lacy Costello, and A. Adamatzky. Evolution of plastic learning in spiking networks via memristive connections. *IEEE Transactions on Evolutionary Computation*, 16:711–729, 2012.

[14] Ella Gale, Ben de Lacy Costello, and Andrew Adamatzky. Memristor-based information gathering approaches, both ant-inspired and hypothetical. *Nano Communication Networks*, 3:203–216, 2012.

[15] Ella Gale, Dave Pearson, Stephen Kitson, Andrew Adamatzky, and Ben de Lacy Costello. The memory-conservation model of memristance. In *Technical Digest of Frontiers in Electronic Materials*, pages 538–539. Nature Conference, Wiley-VCH, June 2012.

[16] Ella M Gale, Benjamin de Lacy Costello, and Andy Adamatzky. The effect of electrode size on memristor properties: An experimental and

theoretical study. In *2012 IEEE International Conference on Electronics Design, Systems and Applications (ICEDSA 2012)*, Kuala Lumpur, Malaysia, November 2012.

Unconventional Computation and Teaching - Proposal for MUSIC, a Tone-Based Scripting Language for Accessibility, Computation and Teaching

[Abbreviated Title: Proposing MUSIC, Tone-input Software Language]

Alexis J. Kirke, Eduardo R. Miranda

Interdisciplinary Centre for Computer Music Research, School of Humanities, Music and Performing Arts, University of Plymouth, Drake Circus, Plymouth, PL4 8AA, UK
Alexis.Kirke@Plymouth.ac.uk

Abstract. *This paper provides a proposal for a tone-based programming/scripting language called MUSIC (the name is an acronym for Music-Utilizing Script Input Code). In a MUSIC program input and output consists entirely of musical tones. Computation can be done through musical transformations of notes and melodies. MUSIC can be used for teaching the basics of script-based programming, computer-aided composition, and provided programming access to those with limitations in sight or physical accessibility. As a result of MUSIC's approach to tone-based programming and computation, it also allows for a development environment that utilizes computer expressive performance for highlighting structure, and emotional transformation to highlight bugs.*

Keywords: Software Development, Whistling Languages, Human-Computer Interaction, Education, Music, Emotion, Computation

1 INTRODUCTION

In this paper a new scripting language is proposed called MUSIC, standing for Music-Utilising Script Input Code. MUSIC is a language whose elements and keywords consist of groups of tones or sounds, separated by silences. The tones can be entered into a computer by whistling, humming, or “LA-LA”ing. Non-pitched sounds can be generated by clucking, tutting or tapping the table top for example. The keywords in MUSIC are made up of a series of non-verbal sounds, sometimes with a particular pitch direction order. MUSIC utilizes a simple script programming paradigm. It does not utilize the MAX/MSP-type or visual programming approach often used by computer musicians, as its tone-based input is designed to help such programmers access and learn script-based approaches to programming.

The purpose of MUSIC is to fivefold: to provide a new way for teaching script programming for children, to provide a familiar paradigm for teaching script programming for composition to non-technically literate musicians wishing to learn about computers, to provide a tool which can be used by blind adults or children to program, to generate a proof of concept for a hands-free programming language utilizing the parallels between musical and programming structure, and to demonstrate the application of musical emotion and computer expressive performance to software sonification.

Although the primary purpose of MUSIC is in the areas of education, accessibility and computer music, it also utilizes new non-standard computation [1] approaches based on musical transformations. Thus as well as being an application of these methods of non-standard computation, it provides a framework within which to investigate them. The provided in this paper involves a common musical transformation in music composition when a single note in a musical motive is replaced by multiple notes. Familiar examples include Mozart String Quartet K.465, Allegro, and Beethoven Symphony No. 7, Movement 2. This method is utilized as a command in MUSIC. It will be demonstrated how it can be used as a multiplicative computation tool.

The fact that computation with these transformations is possible is what makes MUSIC feasible, and this also makes the creation of innovative development and debugging environments possible too. It will be seen that these environments allow the structure of a MUSIC program to be aurally emphasized using computer expressive musical performance; and also provide a way of highlighting bugs through tempo and key-mode transformations.

2 RELATED WORK

There have been musical languages constructed before for use in general (i.e. non-programming) communication – for example Solresol [2]. There are also a number of whistled languages in use including Silbo in the Canary Islands. There are also whistle languages in the Pyrenees in France, and in Oacaca in mexico [3, 4].

A rich history exists of computer languages designed for teaching children the basics of programming. LOGO [5] was an early example, which provided a simple way for children to visualize their programs through patterns drawn on screen or by a “turtle” robot with a pen drawing on paper. Some teachers have found it advantageous to use music functions in LOGO rather than graphical functions [6].

A language for writing music and teaching inspired by LOGO actually exists called LogoRhythms [7]. However the language is input as text. The language was developed so as to teach non-programming-literate musicians to write scripts. Although tools such as

MAX/MSP already provide non-programmers with the ability to build musical algorithms, their graphical approach lacks certain features that a scripting language such as Java or Matlab provide.

As well as providing accessibility across age and skill levels, sound has been used in the past to give accessibility to those with visual issues. Emacspeak [8] for example makes use of different voices/pitches to indicate different parts of syntax (keywords, comments, identifiers, etc). There are more advanced systems which sonify the Development Environment for blind users [9] and those which use music to highlight errors in code for blind and sighted users [10].

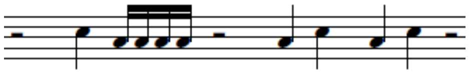
The use of music in unconventional computation can be found in slime-mould-based [11] and in vitro neuron-based [12] synthesizers. In these cases the computations in the substrate are used to generate novel compositional tools. The direct use of music as a computational tool can be found in [13] and [14] in which tunes are used as inputs to a form of "Logic Gate" and "Musical Neuron" to perform emotion-based processing.

3 "MUSIC" INPUT

A MUSIC input string can be an audio file or a MIDI file, consisting of a series of sounds. If it is an audio file then simple event and pitch detection algorithms [15] are used to detect the commands. The command sounds are made up of two types of sound: dots and dashes. A dot is any sound less than 300mS in length, a dash is anything longer. Alternatively a dot is anything less than 1/9 of the longest item in the input stream.

A set of input sounds is defined as a "grouping" if the gaps between the sounds are less than 2 seconds and it is surrounded by silences of 2 seconds or more. Note that these time lengths can be changed by changing the Input Tempo of MUSIC. A higher input tempo setting will reduce the lengths described above. Figure 1 shows two note groupings. The first is made up of a dash and 4 dots, the second grouping is made up of 4 dashes.

Figure 1: Examples of input types



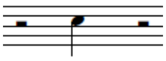
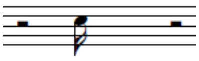
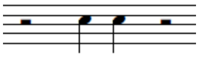
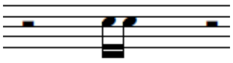
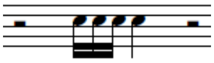
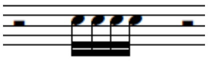
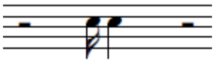
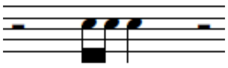
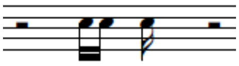
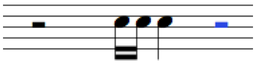
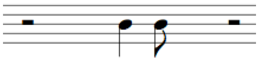
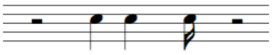
4 COMMANDS

Table 1 shows some basic commands in MUSIC. Each command is a note grouping made up of dots and/or dashes, hence it is surrounded by a rest. The second column gives what is called the Symbol notation. In Symbol notation a dot is written as a period "." and a dash as a hyphen "-". Note grouping gaps are marked by a forward slash "/". The symbol notation is used here to give more insight to those who are unfamiliar with musical notation.

Although MUSIC's commands can be entered ignoring pitch there are pitched versions of the commands which can be useful either to reduce the ambiguity of the sonic detection algorithms in MUSIC or to increase structural transparency for the user. The basic protocol is that a "start something" command contains upward movement or more high pitches, and a "stop something" command contains lower pitches and more downward pitch movement. This can create a cadence-like or "completion" effect (an example of this is shown later).

For example Print could be 4th interval above the End Print pitch. A Repeat command could be two pitches going up by a tone, and End Repeat the same two notes but in reverse pitch order. The rhythm definitions all stay the same and rhythm features are given priority in the sound recognition algorithms on the input in any case. However using the pitched version of MUSIC is a little like indenting structures in C++ or using comments, it is good practice as it clarifies structure. In fact it is possible to change the MUSIC input interface to force a user to enter the pitched version should they wish. In addition it turns a music program into an actual tune, rather than a series of tunes bounded by Morse Code type sounds. This tune-like nature of the program can help the user in debugging (as will be explained later) and to perhaps understand the language from a more musical point of view.

Table 1: Core MUSIC Commands

Input Grouping	**Symbols**	**Name**
	/-/	Print
	/./	End Print
	/--/	Repeat
	/../	End Repeat
	/…-/	Define Object
	/…./	End Object
	/.-/	Use Object
	/..-/	Operator
	/…/	End Operator
	/..--/	Linear Operator
	/-./	Input
	/--./	If Silent

There is also an input mode called Reverse Rhythm, in which the start and stop command rhythms are reversed. In the default input mode shown in Table 1, a command starts with a longer note (a dash), and ends with a shorter note (a dot). However it is quite common in musical cadences to end on a longer note. So it a user prefers they can reverse the rhythms in the stop and start commands in Table 1 by switching to Reverse Rhythm mode.

5 EXAMPLES

The Print command in Table 1 will simply treat the sounds between itself and the Stop Print command as actual musical notes, and simply output them. It is the closest MUSIC has to the PRINT command of BASIC. For example suppose a user approximately whistles, hums and /or clicks the tune shown in Figure 2 (Symbols "/-/BCCD/./"). Then MUSIC will play back the 4 notes in the middle of the figure (B,C,C,D) at the rhythm they were whistled or hummed in.

The Repeat command in Table 1 needs to be followed in a program by a note grouping which contains the number of notes (dots or dashes) equal to the number of repeats required. Then any operation between those notes and the End Repeat note grouping will be repeated that number of times. There are standard repeat signs in standard musical notation, but these are not very flexible and usually allow for only one. As an example of the Repeat command Figure 3 starts with a group of 2 dashes, indicating a Repeat command (Symbols: "/--/…/-/BCCD/./../"). Then a group of 3 dots – indicating repeat 3 times. The command that is repeated 3 times is a Print command which plays the 4 notes at the start of the second line in Figure 3 (B,C,C,D). So the output will be that shown in Figure 4,that motif played three times (BCCDBCCDBCCD).

Figure 2: A Print Example

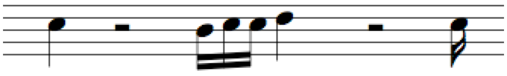


Figure 3: A Repeat Example

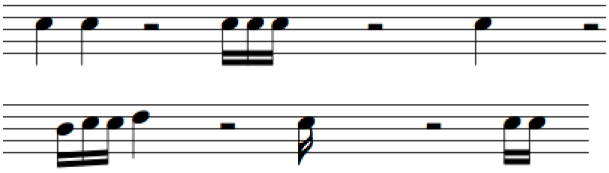
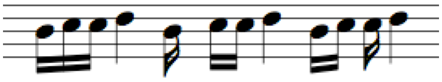


Figure 4: MUSIC Output from Fig. 3 Repeat



5 OBJECTS

The previous example, in Figures 3 and 4, shows a resulting output tune that is shorter than the tune which creates it – a rather inefficient form of programming! Functionality is increased by allowing the definition of Objects. Examples will now be given of an Outputting object and an Operating object. An outputting object will simply play the piece of music stored in it. An example of defining an outputting object is shown in Figure 5. The Define and End Object commands can be seen at the start and end of Fig 5.'s note stream, take from Rows 5 and 6 of Table 1.

The motif in the middle of the top line of Fig. 5 (B,D,B) is the user defined "tone name" of the object, which can be used to reference it later. The contents of the object is the 7 note motif at the start of the second line in Figure 5 (B,C,C,D,C,D,B). It can be seen that this motif is surrounded by Print and End Print commands. This is what defines the object as an Outputting object. Figure 6 shows a piece of MUSIC code which references the object defined in Figure 5. The output of the code in Fig. 6 will simply be to play the tune BCCDCDB twice, through the Use Object command from Table 1.

Figure 5: An Outputting Object

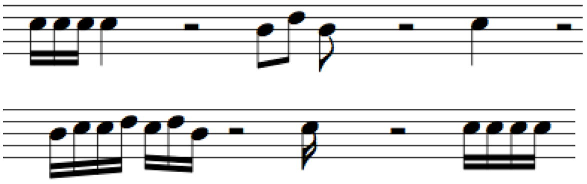
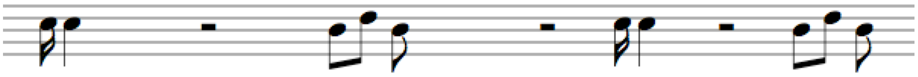


Figure 6: Calling the object from Figure 5 twice



The next type of object - an operating object – has its contents bordered by the Operator and End Operator commands from Rows 8 and 9 in Table 1. Once an operator object has been defined, it can be called, taking a new tune as an input, and it operates on that tune in the common compositional method described earlier in this paper: it can replace each single note by a group of notes. An example is shown in Figure 7.

Figure 7 is the same as Figure 6 except for the use of the Operator and End Operator commands from Table 1, replacing the Print and End Print commands in Fig. 6. The use of the Operator command turns the BCCDCDB motif into an operation rather than a tune. Each pitch of this tune is replaced by the intervals input to the operation. To see this in action consider Figure 8. The top line starts with the Use Object command from Table 1, followed by the name of the object defined in Figure 7. The final part of the top line of Fig. 8 is an input to the operation. It is simply the two notes C and B.

Figure 7: Defining an Operator object

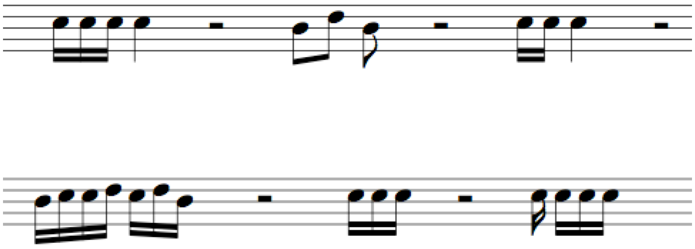
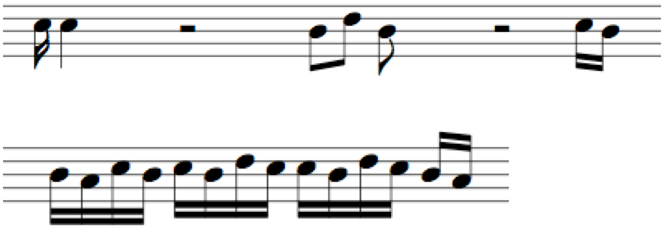


Figure 8: Calling an Operator object, and the resulting output



The resulting much longer output shown in the bottom line of Fig. 8 comes from MUSIC replacing every note in its operator definition with the notes C and B. So its operator was defined in Figure 7 with the note set BCCDCDB. Replacing each of these notes with the input interval CB we get BACBCBDCCBDCBA which is the figure in the bottom line of Figure 8. Note that MUSIC pitch quantizes all data to C Major by default (though this can be adjusted by the user).

5 OTHER COMMANDS AND COMPUTATION

It is beyond the scope of this proposal paper to list and give examples for all commands. However a brief description will be given of the three remaining commands from Table 1.

The Linear Operation command in Table 1 actually allows a user to define an additive operation on a set of notes. It is a method of adding and subtracting notes from an input parameter to the defined operation. When an Input command (in the last-but-one row of Table 1) is executed by MUSIC the program waits for the user to whistle or input a note grouping and then assigns it to an object. Thus a user can enter new tunes and transformations during program execution. Finally the If Silent command in the last row of Table 1 takes as input an object. If and only if the object has no notes (known in MUSIC as the Silent Tune) then the next note grouping is executed.

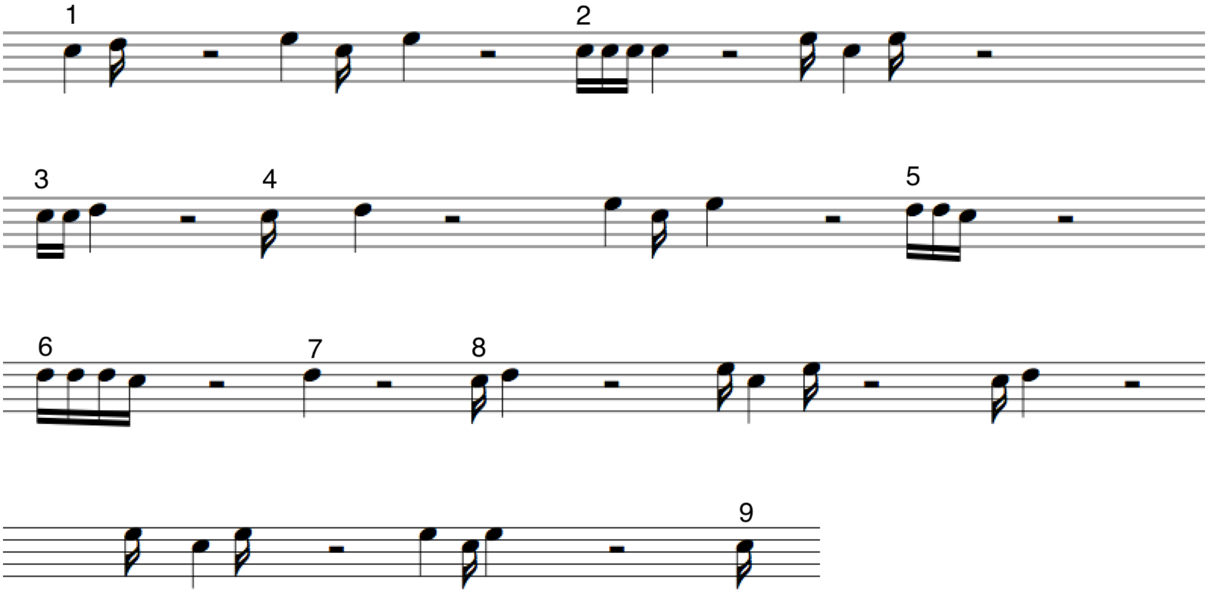
Although MUSIC could be viewed as being a simple to learn script-based "composing" language, it is also capable of computation, even with only the basic commands introduced. For example Printing two tunes T1 and T2 in series will results in an output tune whose number of notes is equal to the number of notes in T1 plus the number of notes in T2. Also, consider an operator object of the type exemplified in Figures 6-8 whose internal operating tune is T2. Then calling that operator with tune T1 will output a tune of length T1 multiplied by T2. Given the Linear Operator command which allows the *removing* of notes from an input tune, and the If Silent command, there is the possibility of subtraction and division operations being feasible as well.

As an example of computation consider the calculation of x^3 - the cube of a number. This is achievable by defining operators as shown in Figure 9. When executed by the user, they can whistle a note grouping of x notes, and have x^3 notes played back. To understand how this MUSIC code works it is shown in pseudocode below. Each line of pseudocode is also indicated in Figure 9.

```
1 Input X
2 Define Object Y
3         Operator
4                 Use Object X
```

```
5       End Operator
6 End Object
7 Print
8       Use Object(Y, Use Object(Y,X)))
9 End Print
```

Figure 9: MUSIC Code to Cube the Number of Notes Whistled/Hummed



Note that MUSIC always auto-brackets from right to left. Hence line 8 of the pseudocode is indeed instantiated in the code shown in Figure 9. Figure 9 also utilizes the pitch-based version of the notation discussed earlier.

6 MUSICO-EMOTIONAL DEBUGGING

Once entered, a program listing of MUSIC code can be done in a number of ways. The musical notation can be displayed, either in common music notation, or in a piano roll notation (which is often simpler for non-musicians to understand). A second option is a symbolic notation such as the Symbols of '/', '.' and '-' in column 2 of Table 1. Or some combination of the words in column 3 and the symbols in column 2 can be used. However a more novel approach can be used which utilizes the unique nature of the MUSIC language. This involves the program being played back to the user as music.

One element of this playback is a feature of MUSIC which has already been discussed: the pitched version of the commands. If the user did not enter the commands with pitched format, they can still be auto-inserted by the development environment and played back in the listing in pitched format - potentially helping the user understand the structure more intuitively.

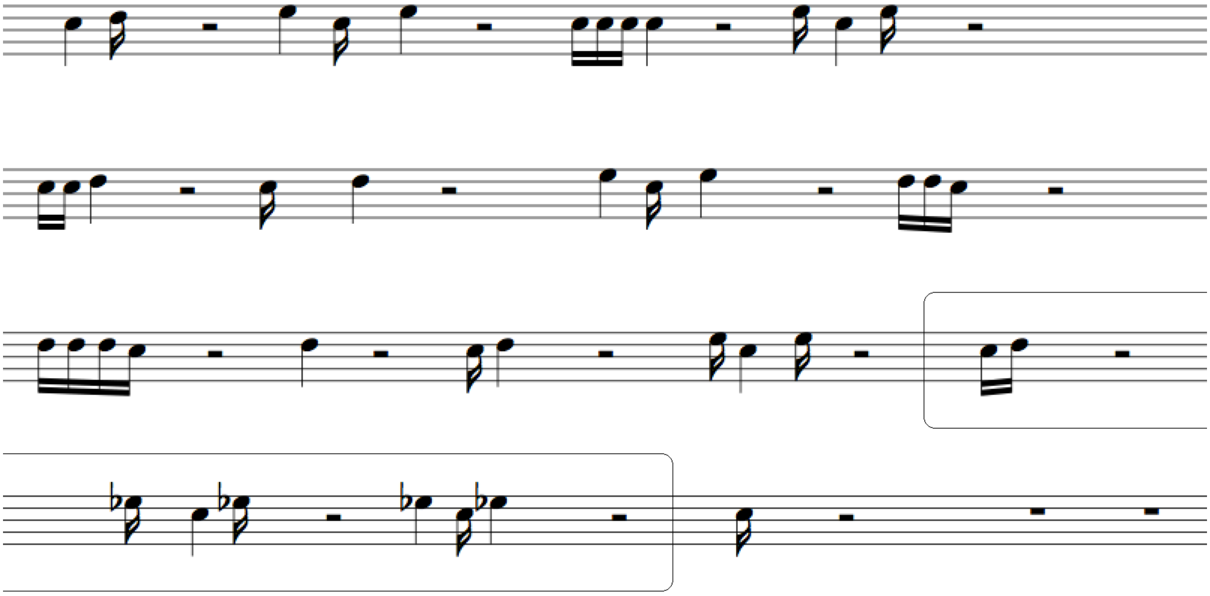
In fact a MUSIC development environment is able to take this one step further, utilizing affective and performative transformations of the music. It has been shown that when a musician performs they will change their tempo and loudness based on the phrase structure of the music composition they're performing. These changes are in addition to any notation marked in the score by the composer. The changes emphasise the structure of the piece [16]. There are computer systems that can simulate this "expressive performance" behaviour [17], and MUSIC utilizes one of these in its debugger. As a result when MUSIC plays back a program which was input by the user, the program code speeds up and slows down in ways not input by the user but which emphasise the hierarchical structure of the code. Like the pitch-based notation this can be compared to the indenting of text computer code.

Figure 9 can be used as an illustration. Obviously there is a rest between each note grouping. However at each of the numbered points (where the numbers represent the lines of the pseudocode discussed earlier) that rest would be played back as a longer rest by the MUSIC development environment, because of the computer expressive music performance. This has the perceptual effect of dividing the program aurally into "groupings of note groupings" as well as note groupings, to the ear of the listener. So what the user will hear is that when the note groupings are related to the same command instantiation, they will be compressed closer together in time – and appear psychologically as a single meta-grouping. Whereas the notes groupings between separate command sections of code (the numbered parts of Figure 9) will tend to be separated by a slightly longer pause. This is exactly the way that musical performers emphasise the structure of a normal piece of music into groupings and meta-groupings and so forth; though the musician might refer to them as motives and themes and sections.

Additionally to the use of computer expressive performance: when playing back the program code to the user, the MUSIC development environment can transform it emotionally to highlight errors in the code. For good syntax the code will be played in a "happy" way – higher tempo and major key. For code with syntax errors, it will be played in a "sad" way – more slowly and in a minor key. Such musical features are known to express happiness and sadness to listeners [18]. The Sadness not only highlights the errors, but also slows down the playback of the code, which will make it easier for the user to understand. Taking the code in Figure 9 as an illustration again, imagine that the user had entered the program with one syntax error, as shown in Figure 10. Four notes in the boxed area have been flattened in pitch (the "*b*" sign) in Figure 10, the reason for which will be explained below.

The note grouping at the start of the highlighted area should have been a 'Use Object' command from Table 1. However by accident the user sang / whistled / hummed the second note too quickly and it turned into an 'End Repeat' command instead. This makes no sense in the syntax, and confuses the meaning of all the note groupings until the end of the boxed area. As a result when music plays back the code it will play back the whole boxed area at two-thirds of the normal tempo. The four notes in the boxed area which have been flattened in pitch (the "*b*" sign) are marked to indicate how the development environment plays back the section of code effected by the error. These flats will turn the boxed area from a tune in the key of C major to a tune in the key of C minor. So the error-free area is played back at full tempo in a major key (a “happy” tune) and the error-affected area is played back at two-thirds tempo in a minor key (a “sad” tune). Not only does this highlight the affected area, it also provides a familiar indicator for children and those new to programming: “sad” means error.

Figure 10: MUSIC Code from Figure 9 with a Syntax Error



7 CONCLUSIONS

A new scripting language called MUSIC has been proposed whose elements and keywords consist of groups of tones or sounds, separated by silences. Computation can be done through musical transformations of notes and melodies. The purpose of MUSIC is fivefold: to provide a new way for teaching script programming for children, to provide a familiar paradigm for teaching script programming for composition to non-technically literate musicians wishing to learn about computers, to provide a tool which can be used by blind adults or children to program, to generate a proof of concept for a hands-free programming language utilizing the parallels between musical and programming structure, and to demonstrate the application of musical emotion and computer expressive performance to software sonification.

It has been demonstrated how MUSIC utilizes new non-standard computation approaches based on musical transformations; and suggested that MUSIC could provide a framework within which to investigate more such transformations. The fact that computation with these transformations is possible is at the heart of what makes MUSIC feasible, and thus what makes the investigation of innovative development and debugging environments for tone-based programming possible. These aural environments highlight the program structure of MUSIC using simulations of human expressive performance, and highlight syntax errors by performance emotional-musical transforms on the code in the areas where the errors occur.

One element of future work in investigating MUSIC is: how well will people who learned script-programming approaches using MUSIC be able to utilize their learned skills in programming languages such as Visual Basic or C, that use standard computation approaches, as opposed to musical transform-based computation? Another key element is investigating the flexibility of the Operator and Linear Operator commands in Table 1 to perform calculations. They can be viewed as roughly analogous to multiplicative and additive calculations. Are there more flexible or powerful musical transformations that can be borrowed from composers and musicologists which can fulfill these functions more efficiently? Or which can extend the practical or theoretical computational power of MUSIC? Investigating these questions will also provide further answers about the utility of musical transforms as a non-standard form of computation.

References

1. Gramb, T., Gram,T. and Pellizzari, T. (1997) Non-Standard Computation, Wiley & Sons, New York, USA
2. Gajewski, B. (1902) Grammaire du Solresol, France
3. Busnel, R.G. and Classe, A. (1976) Whistled Languages. New York: Springer-Verlag.
4. Meyer J. (2005) Typology and intelligibility of whistled languages: approach in linguistics and bioacoustics. PhD Thesis. Lyon University, France
5. Harvey, B. (1998) Computer Science Logo Style, MIT Press, USA
6. Guzdial, M. (1991) Teaching Programming with Music: An Approach to Teaching Young Students About Logo, Logo Foundation, USA
7. Hechmer, A., Tindale, A., Tzanetakis, G. (2006) LogoRhythms: Introductory Audio Programming for Computer Musicians in a Functional Language Paradigm, In *Proceedings of 36th ASEE/IEEE Frontiers in Education Conference*, San Diego, CA, IEEE
8. Raman, T.V. (1996) Emacspeak - A Speech Interface, In *Proceedings of 1996 Computer-Human Interaction Conference*, ACM New York, NY, USA
9. Stefik, A., Haywood, A., Mansoor, S., Dunda, B., Garcia, D. (2009) SODBeans. In *Proceedings of the 17th international Conference on Program Comprehension*. Vancouver, B.C. Canada, IEEE Computer Society
10. Vickers, P., Alty, J.L. (2003) Siren songs and swan songs debugging with music., *Communications of the ACM*, Vol. 46, No. 7, pp. 86-93, ACM New York, NY, USA
11. Miranda, E., Adamatzky, A. and Jones, J. (2011) Sounds Synthesis with Slime Mould of Physarum Polycephalum, Journal of Bionic Engineering 8, pp. 107-113 Elsevier

12. Miranda, E. R., Bull, L., Gueguen, F., Uroukov, I. S. (2009). “Computer Music Meets Unconventional Computing: Towards Sound Synthesis with In Vitro Neuronal Networks”, Computer Music Journal, Vol. 33, No. 1, pp- 9-18.
13. Kirke, A., Miranda, E. (In Press) "Pulsed Melodic Processing – the Use of Melodies in Affective Computations for Increased Processing Transparency". Music and Human-Computer Interaction, S. Holland, K. Wilkie, P. Mulholland and A. Seago (Eds.), London: Springer.
14. Kirke, A., Miranda, E. (2012) Application of Pulsed Melodic Affective Processing to Stock Market Algorithmic Trading and Analysis, Proceedings of 9th International Symposium on Computer Music Modeling and Retrieval (CMMR2012), London
15. Lartillot, O., Toiviainen, P. (2007). MIR in Matlab (II): A Toolbox for Musical Feature Extraction From Audio. In *Proceedings of 2007 International Conference on Music Information Retrieval*, Vienna, Austria.
16. Palmer, C. (1997) Music Performance. *Annual Review of Psychology*, Vol. 48, pp. 115-138, Annual Reviews, Palo Alto, USA
17. Kirke, A., Miranda, E.R. (2012) Guide to Computing for Expressive Music Performance, Springer, USA
18. Lvingstone, S.R., Muhlberger, R., Brown, A.R. (2007) Controlling musical emotionality: An affective computational architecture for influencing musical emotions, Digital Creativity 18(1) pp. 43-53, Taylor & Francis

Self-Assembly of Musical Representations in MGS

LOUIS BIGO[1,2]⋆, ANTOINE SPICHER[1]†

[1] *LACL, University Paris-Est Créteil, FR*
[2] *IRCAM UMR 9912 CNRS - UPMC, FR*

In this paper, we apply morphogenetic processes, namely self-assembly processes, to compute automatically various abstract spaces that can be used to represent and analyze several well-known musical objects (sequence of chords, interval series, etc.). These constructions have been implemented in MGS, an unconventional programming language belonging to the family of spatial computing languages.

Key words: self-assembly, abstract cellular complexes, MGS, rewriting, pattern matching, Hamiltonian and Eulerian path, pitch space, chord space, *tonnetz*, all interval series.

1 INTRODUCTION

The algebraic nature of many musical formalizations has been very early assessed: from the equal temperament to canon, algebraic objects have been used to study combinatorial properties and classify musical structures. Recently, a fresh look on these structures has emerged focusing on topological or geometrical representations. For example, one can characterize harmonic paths in orbifolds [25, 5] or build topological spaces embedding musical relationships in their neighborhood relationships [14].

Following this line of research, we are interested to harness natural morphogenetic processes for building spatial representations of musical objects.

⋆ email: `louis.bigo@ircam.fr`
† email: `antoine.spicher@u-pec.fr`

To this aim, our work rely on the use of MGS, a domain specific programming language dedicated to the modeling and the simulation of *dynamical systems with a dynamical structure* [12]. Numerous applications in system and synthetic biology have been developed in MGS and proved that it is a fruitful unconventional tool for (re-)designing algorithms tackling problems embedded in space or having a spatial extension.

In this paper we propose the use of a self-assembly process for studying two paradigmatic problems in theoretical music. This paper is organized as follows. Section 2 provides a brief introduction to the MGS spatial programming language. Section 2.3 exemplifies the use of the MGS concepts by specifying a self-assembly of polymers. This self-assembly process is then hijacked for musical purpose. In Section 3 a combinatorial space is built up enabling the enumeration and the topological classification of *All-Interval Series* (AIS). Section 4 describes a spatial representation of collections of chords. The paper ends with a conclusion and a discussion about future works.

2 THE MGS PROGRAMMING LANGUAGE

MGS is a *spatial computing* programming language developed to enlighten the importance of space in computations [7, 21, 2]. MGS concepts are based on well established notions in algebraic topology [17] and relies on the use of rule based functions, called *transformations*, to compute declaratively with spatial data structures, called *topological collections*.

2.1 Topological Collections

In MGS, all data structures are unified under the notion of *topological collection*: an *abstract combinatorial complex* (ACC) labeled with arbitrary values. The ACC acts as a container and the labels as the elements of the data structure.

More precisely, an ACC $K = (\mathsf{C}, \prec, [\cdot])$ is a set C of abstract elements, called *cells* [24], provided with a partial order $\prec$ called the *boundary relation*, and with a *dimension* function $[\cdot] : \mathsf{C} \to \mathbb{N}$ such that for each c and c' in C, $c \prec c' \Rightarrow [c] < [c']$. We write $c \in K$ when a cell c is a cell of C.

A cell of dimension p is called a p-cell: 0-cells are points, 1-cells are edges, 2-cells are polygons (*e.g.*, facets in a mesh), etc. For example, a graph is an ACC composed of 0- and 1-cells. Another example is pictured in Figure 1.

The $(p-1)$-cells c' lower than a p-cell c for the boundary relation $\prec$ are called the *faces* of c and we write $c' < c$ or $c > c'$; c is called a *coface* of cells

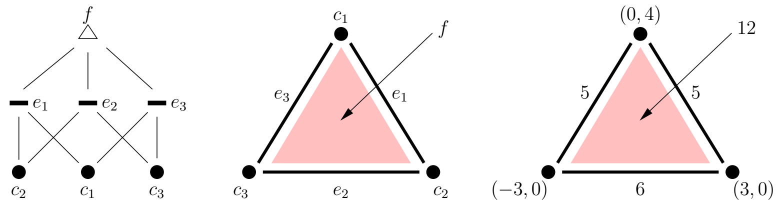


Figure 1
On the left, the Hasse diagram of boundary relationship of the ACC given in the middle: it is composed of three 0-cells (c_1, c_2, c_3), of three 1-cells (e_1, e_2, e_3) and of a single 2-cells (f). The three edges are the faces of f, and therefore f is a common coface of e_1, e_2 and e_3. On the right, a topological collection associates data with the cells: positions with vertexes, lengths with edges and area with f.

c'. We call *closure* of c in K the sub-complexes $\bar{c} = (\mathsf{C}', \prec \cap\, \mathsf{C}' \times \mathsf{C}', [\cdot])$ where $\mathsf{C}' = \{c' \in \mathsf{C} \,|\, c' \preceq c\}$.

Two n-cells are (n,p)-*neighbor* if they have a common border of dimension p when $p \leq n$ or if they are in the boundary of a p-cell of higher dimension. A (n,p)-*path* is a sequence of cells such that two consecutive cells are (n,p)-neighbor. For example, the notion of $(0,1)$-path coincides with the usual notion of path in a graph (a sequence of nodes following a route along some edges of the graph.) We call a (n,p)-*Hamiltonian* path a (n,p)-path visiting each n-cell of an ACC exactly once. Similarly a (n,p)-*Eulerian* path is a (n,p)-path visiting each p-cell of an ACC exactly once.

Finally, a *topological collection* C is a function that associates a value with a cell in an ACC, see Figure 1. Thus the notation $C(c)$ refers to the value of C on cell c. We write $|C|$ for the set of cells for which C is defined. The collection C can be written as a formal sum $\sum_{c \in |C|} v_c \cdot c$ where $v_c \stackrel{\text{df}}{=} C(c)$. With this notation, the underlying ACC is left implicit but can usually be recovered from the context. By convention, when we write a collection C as a sum $C = v_1 \cdot c_1 + \cdots + v_p \cdot c_p$, we insist that all c_i are distinct. Notice that this addition is associative and commutative. This notation is directly used in MGS to build new topological collections on arbitrary ACC of any dimension.

2.2 Transformations

Topological collections are transformed using sets of rules called *transformations* [22]. A rule is a pair `pattern => expression`. When a rule is applied on a topological collection, a sub-collection matching with the `pattern` is replaced by the topological collection computed from `expression`. There exist several ways to control the application of a set of rules on a collection called *rule application strategies*. The present work uses solely the *maximal-parallel* strategy: rules are applied as many times as possible without intersection between matched sub-collections.

A formal specification of topological rewriting is given in [22]. We only sketch here the part of the patterns language necessary for the comprehension. A *pattern variable* specifies a cell to be matched in the topological collection together with some optional guard: the expression `x / x = 3` matches a cell labeled with the value 3. The guard is the predicate after the symbol `/`. The variable `x` can be used in the guard (and elsewhere in the rule) to denote the value of the matched cell or the cell itself, following the context (in case of ambiguity, the variable always denotes the associated value). A pattern is a *composition* of pattern variables. The composition denoted by a simple juxtaposition (e.g., "`x y`") does not constraint the arguments of the composition. Variables can be composed using the (co)face operator: a pattern "`x < y`" (resp. "`x > y`") matches two cells $c_{\mathtt{x}}$ and $c_{\mathtt{y}}$ such that $c_{\mathtt{x}} < c_{\mathtt{y}}$ (resp. $c_{\mathtt{x}} > c_{\mathtt{y}}$). Patterns are *linear*: two distinct pattern variables refer to two distinct cells.

2.3 Illustration: Self-Assembly of Cellular Complexes

MGS is a vehicle used to investigate the notions of topological collections and transformations and to study their adequacy to the simulation of various physical, chemical and biological processes [9, 10, 11, 15]. As an example, (local) rewriting rules are particularly adequate to specify self-assembling processes since they mimic closely the incremental and distributed building mechanism of the real phenomenon.

Let illustrate this idea by considering a generic self-assembling process on cellular complexes allowing the building of elaborated spatial structures from a population of basic elements. This process consists in identifying topologically equivalent elements (*i.e.*, cells with the same boundary) in some ACC. This operation is not elementary because the identification must occur at every dimension. A simple way to achieve such computation is to iteratively apply the merge of topological cells that exactly share the same faces until a fixed point is reached. The corresponding topological surgery can be expressed in the MGS syntax as follows:

```
trans Self-Assembly[Pred, Label] = {
   x y / (Pred x y) and (faces x = faces y)
      => let c = new_cell (dim x) (faces x)
                      (union (cofaces x) (cofaces y))
         in  (Label x y) * c
}
```

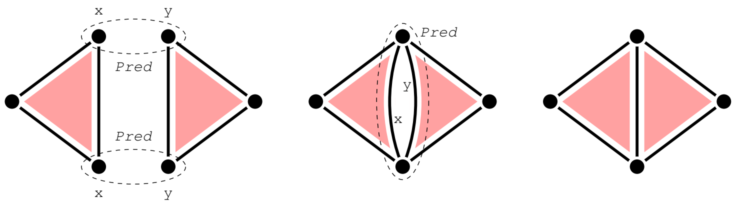


Figure 2
Self-assembly of cellular complexes using transformation `Self-Assembly`: at first iteration, two matching pair of nodes are merged; then the resulting edges are identified; finally the fixed point is reached.

where the primitive `new_cell` $p\ f\ cf$ returns a fresh p-cell with faces f and cofaces cf. The rule specifies that two elements `x` and `y` whose labels check some arbitrary predicate `Pred` (given as a parameter) and having the same faces in their boundaries, merge into a new cell `c` (that has the union of the cofaces of `x` and `y` as cofaces) labeled by a value computed from some arbitrary function `Label` (given as a parameter). Figure 2 illustrates the process.

Figure 3 illustrates the use of transformation `Self-Assembly` to model a polymerization process. Polymers are long-chained molecules with repeating units called monomers. Monomers react with each other to form polymers in a process called polymerization. Addition polymerization is a particular class of polymerization where monomers combine with polymers in an accretive growth process. Here we consider monomers with two binding sites. A monomer is represented by a rectangular shaped ACC, that is a 2-cell with four edges and four vertices in its boundary. The binding sites correspond to two opposite edges labeled as active. Considering that two active cells can be merged (predicate `Pred`) and that they become inactive after merging (function `Label`), transformation `Self-Assembly` builds up polymers as shown in Figure 3. This toy model does not refer to any natural phenomenon but it could be easily extended to manage with real biological data (*e.g.*, modeling the 3-dimensional structure of some DNA strand.)

3 SPATIAL INTERPRETATION OF ALL-INTERVAL SERIES

In this section we are interested in the very basics musical notions of *pitch* and *interval*. Some 2D ACC is then elaborated by the self-assembly of spatial representation of the twelve *interval classes*. The combinatorial structure of this space is finally used to enumerate and classify *All-Interval Series* (AIS).

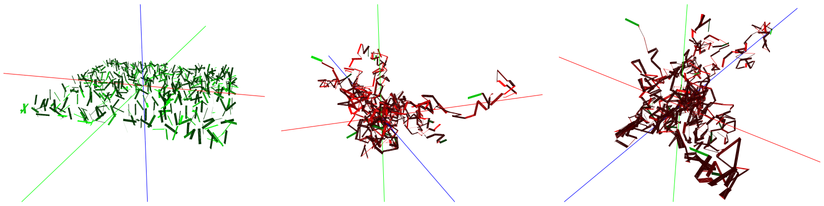


Figure 3
Self-assembly of polymers: on the left, the initial population of monomers; on the right, the final polymer seen from two different points of view.

3.1 Backgrounds in Music Theory

In the present work, we are interested in symbolic representation of music. In standard Western music, the usual notation is based on the concept of staff where notes are represented in two dimensions* : vertical height is associated with the *pitch* of the notes (*i.e.*, how high is the associated sound) and horizontal ordering corresponds to time flow (*i.e.*, when notes have to be played.)

In musical analyses, pitches are often considered up to an octave letting us work with only twelve classes called *pitch classes*. For example, the pitch class $C = \{C_0, C_1, C_2, \dots\}$ gathers all the possible Cs. From now on, the term "note" will refer to pitch classes. It is then usual to identify pitch classes to elements of $\mathbb{Z}_{12}$ ($C = 0$, $C\sharp = 1$, etc.) such that the difference modulo 12 between two pitch classes exactly corresponds to the number of semitones between the corresponding two notes. All the possible differences constitute the *intervals*. For example between $G = 7$ and $D = 2$, the interval is a *perfect fifth* corresponding to $2 - 7 = -5 \equiv (7 \mod 12)$ semitones. In the following, intervals are refered using the usual notation: $P1 = 0$ (*perfect union*), $m2 = 1$ (*minor second* or semitone), $M2 = 2$ (*major second*), $m3 = 3$ (*minor third*), $M3 = 4$ (*major third*), $P4 = 5$ (*perfect forth*), $TT = 6$ (*tritone*), $P5 = 7$ (*perfect fifth*), $m6 = 8$ (*minor sixth*), $M6 = 9$ (*major sixth*), $m7 = 10$ (*minor seventh*), $M7 = 11$ (*major seventh*).

The identification with intergers of $\mathbb{Z}_{12}$ provides the set of intervals with an additive group structure, and the natural action of $(\mathbb{Z}_{12}, +)$ on itself coincides with the *transposition* operation in music. As an example, the action of the perfect forth transpose A into $A + P4 = 9 + 5 \equiv (2 \mod 12) = D$. The action of each interval on the notes is composed of a variable number of orbits. For an interval i it is well known that the number of orbits is $d_i =$

---

* We drop here the consideration of duration.

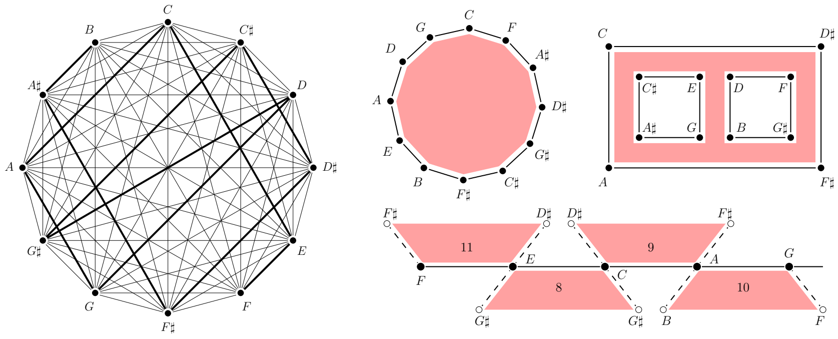


Figure 4
Spatial representation of AIS. On the left, complete graph topology with A. Berg's AIS in bold. On the top right, spatial representation of the perfect fourth and minor third interval classes (2-cells are filled in light gray). On the bottom right, spatial representation using interval classes of the five first elements of A. Berg's AIS.

$\gcd(i, 12)$. For instance, there are $d_{m3} = \gcd(3, 12) = 3$ orbits for the minor third:

$$(C - D\sharp - F\sharp - A) \quad (C\sharp - E - G - A\sharp) \quad (D - F - G\sharp - B)$$

These cycles can be uniquely identified by an integer between 0 and $d_i - 1$ corresponding to the least note of the cycle (*e.g.*, C, $C\sharp$ and D for the three cycles above).

3.2 Spatial Representation of Interval Classes

In this section, we propose to give a combinatorial construction of the previous formal elements. A usual representation consists of the *all-interval circle* which is a complete graph with twelve nodes, one for each pitch class. Each edge represents the possible transposition from a note to another under the action of an interval. This structure is represented on left of Figure 4. The all-interval circle has two main drawbacks: (1) it does not distinguish intervals and their inverses (*e.g.*, the edge between C and F represents at the same time a perfect fourth a perfect fifth); (2) each interval is represented many times (one for each note).

We propose to elaborate a more complex structure which exhibits these properties by assembling a population of pieces of space called *interval classes*. The interval class I_i is a topological collection representing interval i. The underlying ACC is composed of a unique 2-cell labeled by i. The faces of this cell include twelve 1-cells also labeled by i. Finally twelve 0-cells labeled by the pitch classes constitute the faces of the 1-cells in such a way that two vertices are connected by an edge if their labels are related by the action of i. Top right of Figure 4 illustrates the interval classes associated to perfect fourth and minor third. Note that the boundary of an interval class exibits topologically the orbits of the interval: the number of orbits equals the number of holes + 1.

The eleven interval classes – perfect unison is not considered – are then glued together by transformation `Self-Assembly` which will identify the note (only 0-cells are concerned) with

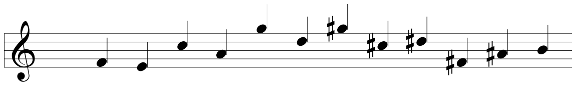
```
Pred  x y = (x = y)
Label x y = x
```

The resulting space is not easily visualizable. Nevertheless, its 1-skeleton (that is the the sub-ACC composed of the n-cells with $n \leq 1$) coincides with the all-interval circle.

3.3 Application to All-Interval Series

The interval class space has interesting properties which allow the study of musical objects from the point of view of combinatorial topology. It is illustrated in next paragraphs with a study of *All-Interval Series* (AIS).

All-Interval Series. An AIS is a twelve-tone series including the eleven different intervals. Such series exhibit many notable properties [16]. One of them is that the first and the last notes are always separated by a tritone interval. One of the most famous use of this particular kind of series is probably on the *Lyric Suite* of Alban Berg:



Notes: 5 4 0 9 7 2 8 1 3 6 10 11
Intervals: 11 8 9 10 7 6 5 2 3 4 1

Other composers like Luigi Nono (e.g., *Il canto sospeso*) or Karlheinz Stockhausen (e.g., *Grüppen*, *Klavierstück IX*) used this material in their compositions.

Beyond the simple enumeration of the 46 272 AIS, composers and music analysts have been interested in finding relevant criterions to classify them. André Riotte proposed a classification considering the harmonic content of the AIS. It consists for example in grouping together AIS containing a subsequence corresponding to the notes of particular scales or chords [19]. Elliot Carter investigated a classification enumerating all AIS containing in sequence the complete set of notes included in the All-Triad Hexachord (this 6-chord is the only one containing the twelve possible triads) [20]. We can also mention an original classification from Franck Jedrzejewski based on knot theory [13].

The enumeration and the classification of AIS is a widely known problem in the computing music community. Several computing approaches, more and more optimized, have been used to enumerate all the AIS. One of the first enumeration was done by André Riotte [19] with the help of a FORTRAN program. This enumeration problem has quickly become a classical problem in Constraint Programming [23] and is now part of the 50 problems of the CSPLib [8]. Some previous works have been based on the enumeration of the All-Interval Chords, which is a similar problem [18].

Spatial Structure of AIS. A naive procedure to collect all the AIS consists in enumerating all the possible permutations of the twelve notes and keeping those exibiting the eleven different intervals. A spatial interpretation of permutation is to look for an Hamiltonian path in the all-interval circle as illustrated in Figure 4 with A. Berg's AIS.

With such a search the $46\,272$ AIS are lost in the set of 12! candidates. $(0,1)$-Hamiltonicity is not a sufficient condition for specifying AIS. An interesting feature of AIS is they are at the same time a permutation of notes and a permutation of intervals. Translated in structural terms, an AIS is a path of the interval class space which is at the same time $(0,1)$-Hamiltonian and $(0,2)$-Eulerian. It is characterized in a `MGS` pattern as follows:

```
n0 < i1  < I1  > i1  > n1
   < i2  < I2  > i2  > n2
   ...
   < i11 < I11 > i11 > n11
```

Two consecutive notes `n`p' and `n`p with $p' = (p-1)$ have to be $(0,1)$-neighbors by some interval `i`p and $(0,2)$-neighbors by some interval class `I`p such that the interval `i`p belongs to (*i.e.*, is in the boundary of) interval class `I`p. Linearity of patterns ensures Hamiltonicity. Figure. 4 on bottom right illustrates the instantiation of this pattern.

Topological Classification of AIS. Each AIS visits only one 1-cell in the boundary of each 2-cell and this 1-cell belongs to one of the different orbits associated with the interval class. Let then consider the *cyclic vector* $\mathbf{V} = (v_{m2}, \dots, v_i, \dots, v_{M7})$ of an AIS which associates the visited cycle v_i with each interval i. For example, the cyclic vector $\mathbf{V}_B$ of the A. Berg's AIS is

interval class i	$m2$	$M2$	$m3$	$M3$	$P5$	TT	$P5$	$m6$	$M6$	$m7$	$M7$
d_i	1	2	3	4	1	6	1	4	3	2	1
$\mathbf{V}_B$	C	$D\sharp$	C	D	C	D	C	C	C	$C\sharp$	C

AIS can then be classified by gathering in the same class all the AIS sharing the same cyclic vector. This classification has proved to be of great musical interest [4]. For instance, it allows to find AIS including some specific subsequence of notes [19, 20]. As an example, the interval content of the harmonic C minor scale $C\ D\ E\flat\ F\ G\ A\flat\ B$ invites us to search for some interesting AIS related to that scale with cyclic vector $(C, C\sharp, D, E, C, F, C, C, C\sharp, C, C)$. For example, this class contains the AIS

$$E \quad D\flat \quad G\flat \quad \boxed{F \quad B \quad D \quad C \quad A\flat \quad E\flat \quad G} \quad A \quad B\flat$$

which exhibits consecutively the seven notes of the scale.

Finally, the proposed classification also allows an easy study of AIS up to the standard algebraic operations [16]: transposition, homothety, retrograde and circular shift. The geometry of the $\prod_i d_i = 3\,456$ possible classes can be folded into a smaller space of only 72 well identified classes.

4 SPATIAL REPRESENTATION OF CHORD COLLECTIONS

Chords play an important role in music theory. In this section, we propose to use the self-assembly process of Section 2.3 to build a combinatorial space representing some collections of chords. We first describe this construction, then we show how various spaces investigated in music theory are recovered.

4.1 Self-Assembly of Chords

A *chord* is a collection of pitches played simultaneously. Depending on the context the collection may have different structures: a set of pitches (*e.g.*, an event of a staff), a sequence of pitches (*e.g.*, in a choral), an ordered set of pitch classes (*e.g.*, chord progressions in Jazz), etc. In the following we will consider chords as set of pitch classes and we call p-chord a chord composed of p notes ($p > 0$).

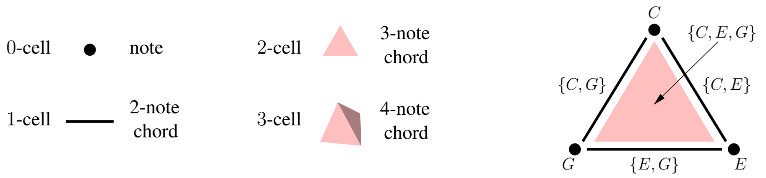


Figure 5
A chord represented as a simplex. The complex on the right corresponds to first degree I_C of the C major tonality and all 2-chords and notes included in it.

Chords as Simplicial Complexes. Since a p-chord includes $2^p - 2$ sub-chords, a p-chord can be represented by a topological collection relying on a $(p - 1)$-simplex. A *n-simplex* is an ACC of dimension n composed of a unique n-cell c which has exactly $n + 1$ faces such that for each face c' the closure of c' is a $(n - 1)$-simplex. The 0-cells are labeled by a single note and other p-cells by the corresponding $(p + 1)$-chords. Simplices are often represented geometrically as the convex hull of their vertices as shown in Figure 5 for p-simplices with $p \in \{0, 1, 2, 3\}$.

Self-Assembly of Chords. Transformation `Self-Assembly` may act on a given set of chords by identifying n-cells with common labels:

```
Pred  x y = (x = y)
Label x y = x
```

Although these parameters are similar to the previous example, the identification occurs here at any dimension and will result in a *simplicial complex* (*i.e.*, an ACC made of simplices only).

The following sections give three applications of self-assembly of chords.

4.2 Analysis of a Musical Piece

The self-assembly process is generic enough to represent any set of chords. We propose here to elaborate some space for the study of a piece from its harmonic progression.

Figure 6 shows the simplicial complex resulting from the assembly of chords from the eight first measures of Chopin's Prelude 4 Op.28. The complex exhibits neighborhoods between chords but does not give any information about how these chords are ordered in the prelude. A remarkable fact of this ordering is that only one note differs between two consecutive chords. This property holds on the fourteen chords starting from the second one.

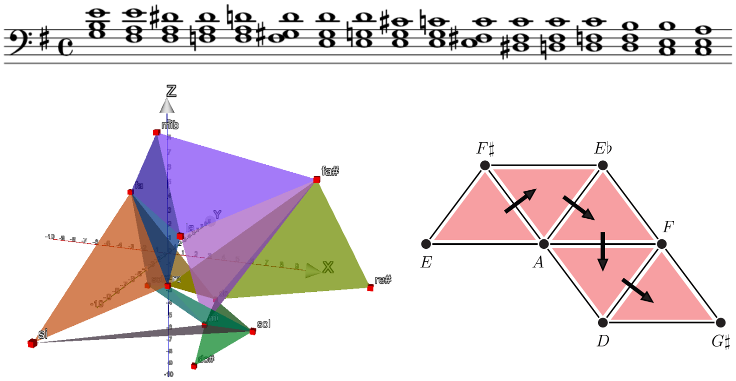


Figure 6
Chords of height first measures of Chopin's Prelude 4 Op.28. On the left its simplicial representation. On the right, a path represents the order of chords in a region of the complex.

Being composed of three-note chords, such a progression corresponds to a $(2,1)$-Hamiltonian path in the associated simplicial complex: 1-cells neighborhood between two 2-cells represents the two common notes between the chords. This path is partially presented by black arrows for the five first chords, starting from the second one, in Figure 6. The enumeration of all the possible $(2,1)$-Hamiltonian paths in the complex, shows that there exist exactly 120 possible ordering of the chords. But among all these possibilities, the original order used in the Prelude is the one with the smallest distance between chords. Indeed, the interval characterizing the moving note in two consecutive chords is a semitone for all transitions. This example illustrates the topological translation of a well-known compositional strategy called *parsimonious voice leading*. Chord sequences corresponding to other $(2,1)$-Hamiltonian in the complex have been generated with `MGS` and are available in MIDI format at
`http://www.lacl.fr/~lbigo/aisb13#analysis`.

4.3 Tonality Representation

In tonal music, a piece is frequently characterized by the use of successive tonalities. A tonality is defined by a set of notes, each having a particular role. *Triadic chords* (stacked thirds 3-chords) composed by notes of the tonality are called the *degrees* of the tonality. As an example, the seven degrees of the C

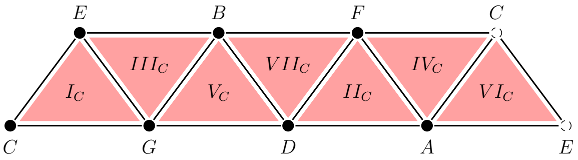


Figure 7
Simplicial representation of C major tonality resulting in a Möbius strip.

major tonality are:

$$I_C = \{C, E, G\} \quad II_C = \{D, F, A\} \quad III_C = \{E, G, B\} \quad IV_C = \{F, A, C\}$$
$$V_C = \{G, B, D\} \quad VI_C = \{A, C, E\} \quad VII_C = \{B, D, F\}$$

The self-assembly provides a combinatorial space associated with the tonality. Guérino Mazzola presents in [14] this topological representation of the diatonic tonality which appears to be a Möbius strip (see Figure 7.)

Such a self-assembly process may seem trivial but the elaboration of a space based on chords of higher dimension is difficult by hand. For example tonality may be defined through 4-chords instead of triads (*e.g.* for C major $I_C = \{C, E, G, B\}$, $II_C = \{D, F, A, C\}$, etc.) After computing the Euler characteristic and the orientability coefficient of the obtained complex for C major characterized by 4-chords, its topology appears to be a *toroid* (the volume bounded by a torus) which is definitively different from a Möbius strip (*e.g.*, the torus is orientable and the Möbius strip is not.)

4.4 Computer Aided Analysis and Composition using Self-Assembled Spaces

In the context of *Music Set Theory* chords are studied from an algebraic point of view [1]. They are then classified as orbits under the action of some group of transformations. We are here interested in the classification induced by the action of the dihedral group.

Chord Classes and Generalized Tonnetze. The dihedral group D_{12} allows to gather chords up to transposition and inversion. While transposition T_i consists of the action of some interval on the notes of a chord as mentioned above, inversion S considers the inverse of the notes. As an example, $(T_{M3} \circ S)\{C, E, G\} = \{-C + M3, -E + M3, -G + M3\} = \{C, E, A\}$. This action defines 224 equivalent classes of chords.

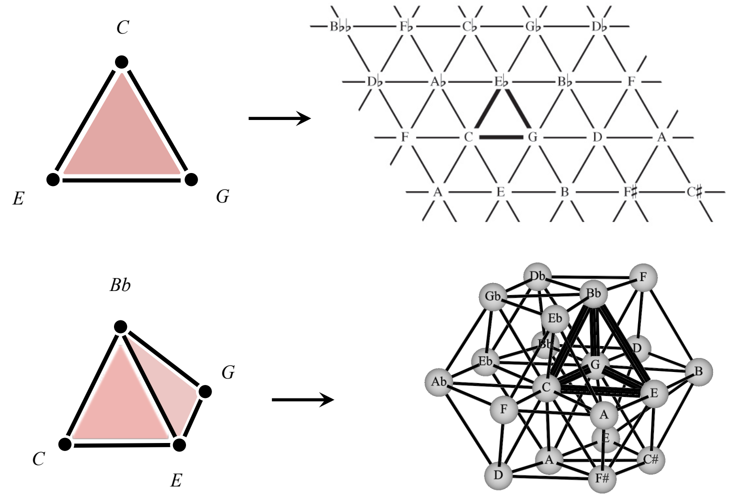


Figure 8
Assembly of chords equivalent up to transposition and inversion to C Major (C, E, G) (top) and to $C7$ $(C, E, G, B\flat)$ (bottom).

Using the self-assembly process, we propose to represent each chord class by a simplicial complex. The topology of these classes differ widely from one to another depending on the size of the considered chords and their interval contents. A complete mathematical study of the topologies for 3-chords can be found in [6]. Figure 8 illustrates *unfolded* representations of the self-assembled structures 1-skeleton for the classes of $\{C, E, G\}$ and $\{C, E, G, B\flat\}$.

A remarkable fact is that the 1-skeletons of chord class spaces are particular graphs known in music theory as *Generalized Tonnetze*. Tonnetz stands for tone network and consists in a regular graph labeled by pitch classes and generated by the action of a subset of intervals. Nevertheless higher dimensional cells of chord class spaces provide additional informations not present in original tonnetze and open new analysis and composition possibilities.

Transformations on musical trajectories. A musical sequence can be represented as a moving object evolving in a space, as the ones illustrated on figure 8. The static representation of this evolution is a trajectory inside the complex, which consists of sub-complexes representing successive played notes. Figure 9 illustrates two trajectories in a chord class space. Discrete counterparts of some euclidean transformations can be applied on such trajectories.

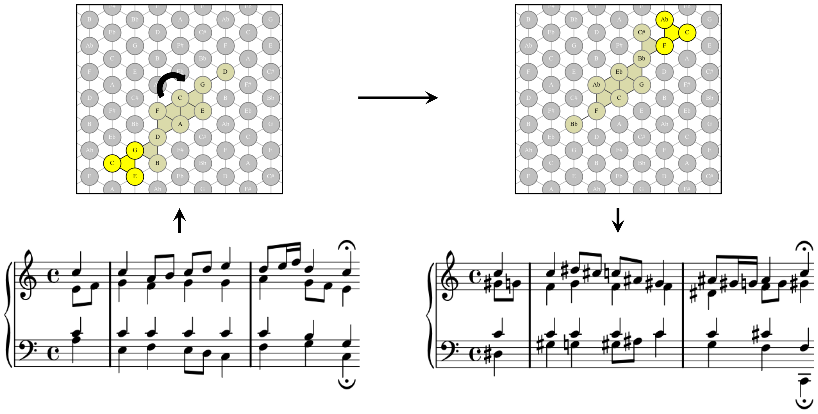


Figure 9
A discrete π rotation is applied on a trajectory representing the first measures of J.-S.Bach's choral BWV 256. The underlying space is the chord class complex built from the assembly of minor and major chords.

Figure 9 illustrates a discrete π rotation applied on a trajectory representing the first measures of J.-S. Bach's choral BWV 256. The underlying space is the chord class complex built from the assembly of minor and major chords. Moreover a spatial transformation can be applied on the underlying space, rather than on the path. These transformations occur on the labels of the elements only, which means that the topological structure of the underlying space stays unchanged. This property ensures that the aspect of the trajectory is conserved.

Some of these spatial transformations have a musical interpretation. For example, a translation leads to a transposition or a modal transposition, depending on the underlying space. A π rotation leads to an musical inversion. Other available transformations lead to alternative musical operations.

Audio results of rotations, translations and transformations of the underlying space, applied to some pieces from various composers (J.-S. Bach, W.A. Mozart, M. Babbit, The Beatles,...) are available in MIDI format at `http://www.lacl.fr/~lbigo/aisb13#transformations`. Example 5 of the online page corresponds to the transformation illustrated on figure 9.

HexaChord and PaperTonnetz Software. The previous theoretical considerations have been implemented in two original software based on the notions of generalized Tonnetz and chord class: HexaChord[†] and PaperTonnetz[‡].

HexaChord is a computer-aided music analysis environment based on the previous spatial representation of chord classes. Beyond the simple visualization of the different 12 2-dimensional simplicial complexes, HexaChord provides musicologists with three main functionalities on imported midi files: (1) HexaChord computes automatically the best Tonnetze for representing some piece; it is based on the computation of a *compliance measure* defined at general level on cellular complexes. (2) Giving a particular Tonnetze, HexaChord is able to produce a trajectory from a piece to show interesting geometric properties. (3) Harmonization by spatial criteria is available by adding an extra voice to a set of voices (*e.g.*, a choral) to maximize compliance measurement.

PaperTonnetz [3] is a tool that lets musicians explore and compose music with Tonnetz representations by making gestures on interactive paper. It allows composers to discover, improvise and assemble musical sequences in a Tonnetz by creating replayable patterns that represent pitch sequences and/or chords. Figure 10 shows GUI of PaperTonnetz and HexaChord.

5 CONCLUSION

A notable contribution of this paper is to show that the same generic self-assembly process can be used to build abstract spaces representing various well-known musical objects: the all-interval series, the simplicial representations of chord sets and the associated Tonnetze.

The generic self-assembly process has been implemented in MGS[¶] providing a formal and very concise specification. The corresponding MGS transformation has been effectively used to compute and enumerate these spaces. Then, topological invariants can be computed to investigate and classify them, especially when they have a high dimension.

Works presented here are related to the unconventional computing field in many aspects. First of all, the different computations presented here are specified using topological rewriting, an extension of term rewriting to abstract cellular complexes. Such extension subsumes chemical computing *à la* Gamma (as commutative-associative rewriting), Lindenmayer systems (as string rewriting) and cellular automata (as a kind of constrained array rewriting). Secondly, the algorithms used are directly inspired by mechanisms in-

---

[†] http://vimeo.com/38102171

[‡] http://vimeo.com/40072179

[¶] `http://mgs.spatial-computing.org`

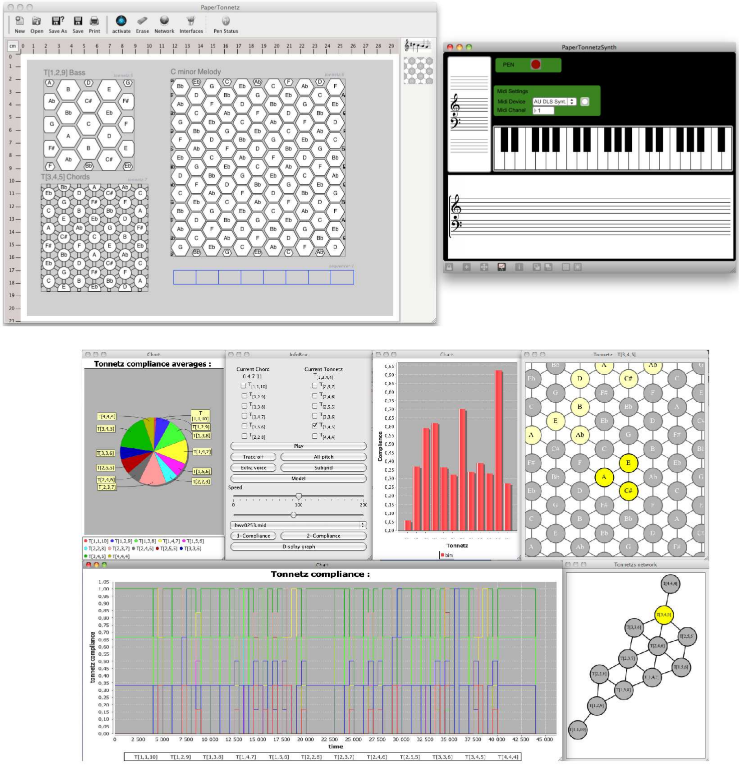


Figure 10
Graphical User Interface of PaperTonnetz (top) and HexaChord (bottom) .

vestigated in the field of unconventional computing. These mechanisms are used as a very effective heuristic to devise new representations of well known musical objects and processes. For instance, we use self-assembly to build a space associated with a set of chords and we use computations in space to represent and extend musical transformations. Last but not least, from a musical perspective the transformations presented here are highly non conventional. As a matter of fact, the topological framework has been used only very recently to formalize musical notions, while the standard approach relies solely on algebraic structure, *e.g.*, for pitch classes in Set Theory. The topological framework renews our point of view on musical problems and suggests some alternative musical transformations. It also suggests new generative processes to produce music, an area not covered in this paper.

We believe that this preliminary work shows the interest of an expressive and generic framework making possible the systematic building and processing of abstract spaces that appear in musical analysis and theory. Our framework is based on spatial notions developed and studied in algebraic topology, and then amenable to a computer implementation due to their algebraic nature. Initially developed for the modeling and the simulation of dynamical systems, it appears well suited for the musicologist.

Designing original interesting spaces is a first step work. We are currently working on unconventional approaches (*e.g.*, cellular automata, random walk algorithms, etc.) for taking advantage of these spaces for compositional purposes.

6 ACKNOWLEDGMENTS

The authors are very grateful to C. Agon, M. Andreatta, G. Assayag, J.-L. Giavitto from the REPMUS team at IRCAM and to O. Michel from the LACL at University of Paris-Est for fruitful discussions. This research is supported in part by the IRCAM and the University Paris Est-Créteil Val de Marne.

REFERENCES

[1] M. Andreatta and C. Agon. (2003). Implementing algebraic methods in openmusic. In *Proceedings of the International Computer Music Conference, Singaphore*.

[2] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll. (2012). Organizing the aggregate: Languages for spatial computing. *arXiv preprint arXiv:1202.5509*.

[3] L. Bigo, J. Garcia, A. Spicher, W.E. Mackay, *et al.* (2012). Papertonnetz: Music composition with interactive paper. In *Sound and Music Computing*.

[4] L. Bigo, J.L. Giavitto, and A. Spicher. (2011). Building topological spaces for musical objects. *Mathematics and Computation in Music*, pages 13–28.

[5] C. Callender, I. Quinn, and D. Tymoczko. (2008). Generalized voice-leading spaces. *Science*, 320(5874):346.

[6] M.J. Catanzaro. (2011). Generalized tonnetze. *Journal of Mathematics and Music*, 5(2):117–139.

[7] André De Hon, Jean-Louis Giavitto, and Frédéric Gruau, editors. (3-8 september 2006). *Computing Media and Languages for Space-Oriented Computation*, number 06361 in Dagsthul Seminar Proceedings. Dagsthul, `http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=2006361`.

[8] I. P. Gent, T. Walsh, I. Hnich, and I. Miguel, (accessed in january 2011). CSPLib: a problem library for constraints. web page at `http://www.csplib.org`.

[9] J.-L. Giavitto and O. Michel. (2003). Modeling the topological organization of cellular processes. *BioSystems*, 70(2):149–163.

[10] Jean-Louis Giavitto. (2003). Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In *Rewriting Technics and Applications (RTA'03)*, volume 2706 of *Lecture Notes in Computer Science*, pages 208–233, Valencia. Springer.

[11] Jean-Louis Giavitto and Antoine Spicher. (2008). *Systems Self-Assembly: multidisciplinary snapshots*, chapter Simulation of self-assembly processes using abstract reduction systems, pages 199–223. Elsevier. doi:10.1016/S1571-0831(07)00009-3.

[12] J.L. Giavitto, O. Michel, and A. Spicher. (2011). Interaction based simulation of dynamical system with a dynamical structure (ds) 2 in mgs. In *Proceedings of the 2011 Summer Computer Simulation Conference*, pages 99–106. Society for Modeling & Simulation International.

[13] Franck Jedrzejewski. (2006). *Mathematical Theory of Music*. Delatour.

[14] G. Mazzola *et al.* (2002). *The topos of music: geometric logic of concepts, theory, and performance*. Birkhäuser.

[15] Olivier Michel, Antoine Spicher, and Jean-Louis Giavitto. (December 2009). Rule-based programming for integrative biological modeling – application to the modeling of the λ phage genetic switch. *Natural Computing*, 8(4):865–889.

[16] Robert Morris and Daniel Starr. (1974). The structure of all-interval series. *Journal of Music Theory*, 18(2):pp. 364–389.

[17] James Munkres. (1984). *Elements of Algebraic Topology*. Addison-Wesley.

[18] Thorvald Otterström. (1935). *A theory of Modulation*. Da Capo press, Inc.

[19] André Riotte and Marcel Mesnage. (2006). *Formalismes et modèles musicaux*. Delatour.

[20] David Schiff. (1983). *The Music of Elliott Carter*. Faber and Faber.

[21] SCW, (accessed in january 2011). the "Spatial Computing Workshops" series and related events. List on the "Spatial Computing Home Page" `http://www.spatial-computing.org/doku.php?id=events:start`.

[22] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. (September 2010). Declarative mesh subdivision using topological rewriting in mgs. In *Int. Conf. on Graph Transformations (ICGT) 2010*, volume 6372 of *LNCS*, pages 298–313.

[23] C. Truchet and P. Codognet. (2004). Musical constraint satisfaction problems solved with adaptive search. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 8:633–640.

[24] A.W. Tucker. (1933). An abstract approach to manifolds. *Annals of Mathematics*, 34(2):191–243.

[25] D. Tymoczko. (2006). The geometry of musical chords. *Science*, 313(5783):72.

1

Towards Harmonic Extensions of Pulsed Melodic Affective Processing – Further Musical Structures for Increasing Transparency in Emotional Computation

[Abbreviated Title: Towards PMAPh - PMAP with Harmony]

Alexis J. Kirke, Eduardo R. Miranda

Interdisciplinary Centre for Computer Music Research, School of Humanities, Music and Performing Arts, University of Plymouth, Drake Circus, Plymouth, PL4 8AA, UK
Alexis.Kirke@Plymouth.ac.uk

Abstract. *Pulsed Melodic Affective Processing (PMAP) is a method for the processing of artificial emotions in affective computing. PMAP is a data stream which can be listened to, as well as computed with. The affective state is represented by numbers which are analogues of musical features, rather than by a binary stream. Previous affective computation has been done with emotion category indices, or real numbers representing positivity of emotion, etc. PMAP data can be generated directly by sound (e.g. heart rates or key-press speeds) and turned directly into into music with minimal transformation. This is because PMAP data is music and computations done with PMAP data are computations done with music. Why is this important? Because PMAP is constructed so that the emotion which its data represents at the computational level, will be similar to the emotion which a person "listening" to the PMAP melody hears. So PMAP can be used to calculate "feelings" and the result data will "sound like" the* *feelings calculated. Harmonic PMAP (PMAPh) is an extension of PMAP allowing harmonies to be used in calculations.*

Keywords: Communications, Human-Computer Interaction, Music, Affective Computing, Boolean Logic, Neural Networks, Emotions, Multi-Agent Systems, Robotics, Harmony, Computation

1 INTRODUCTION

Previous work on unconventional computing and music has focused on using unconventional computation methods as engines for new modes of musical expression. For example using in vitro neural networks [1] or slime molds [2] to drive a sound synthesizer. The research has not focused on studying the computational properties of the underlying unconventional computation, but on examining the feasibility of the sound generation for compositional purposes, and of applying some control to the generation. In the current paper, music is utilized not as an output of unconventional computation but as a method of unconventional computation. It has been said that almost any set of symbols with some structural interaction rules can be used to compute. However music holds a unique place as a form of emotional expression [3]. Thus if the symbols and rules of music can be used in computations related to affective (emotional) processes, then the possibility arises of a symbol set which expresses the results of its own computational result.

This paper is based on the idea of using melodies as a tool for affective computation in artificial systems. Such an idea is not so unusual when one considers the data stream in spiking neural networks. Spiking neural networks (SNNs) have been studied both as artificial entities and as part of biological neural networks in the brain. These are networks of biological or artificial neurons whose internal signals are made up of spike or pulse trains that propagate through the network in time. [4] has actually developed a back-propagation algorithm for artificial SNNs. Back-propagation is one of the key machine learning algorithms used to develop neural networks which can respond intelligently. In SNN's it is usually the spike rate (or "tempo") that encodes information, not the spike height, though there has been some biological work suggesting spike height, in addition to spike rate, may encode information in SNNs in the brain [5,6]. It is interesting to note that a series of timed pulses with differing heights can be naturally encoded by one of the most common musical representations used in computers: MIDI (the musical instrument digital interface). In its simplest form MIDI encodes a melody, which consists of a note timing and note pitch height. In this paper it is argued that melodies can be viewed as functional as well as recreational – they can fulfill the function of encoding an artificial emotional state in a form which can be used in affective computation directly expressible to human beings (or indeed to other machines). The basis of the data stream used in this paper for processing is a pulse stream in which the pulse rate encodes a tempo, and the pulse height encodes a pitch. This is extended here to include the use of harmonies to encode and compute affectivity.

1.1 Uses and Novelty of PMAP

Before explaining the motivations behind PMAP in more detail, an overview of the useful functionality will be given. Similarly the novel functionality which PMAP provides will be summarized. In terms of functionality PMAP provides a method for the processing of artificial emotions, which is useful in affective computing – for example combining emotional readings for input or output, making decisions based on that data or providing an artificial agent with simulated emotions to improve their computation abilities. In terms of novelty, PMAP is novel in that it is a data stream which can be listened to, as well as computed with. The affective state is represented by numbers which are analogues of musical features, rather than by a binary stream of 1s and 0s. Previous work on affective computation has been done with normal data carrying techniques – e.g. emotion category index, a real number representing positivity of emotion, etc.

This element of PMAP provides an extra utility – PMAP data can be generated directly by sound and turn directly into sound. Thus rhythms such as heart rates or key-press speeds can be directly turned into PMAP data; and PMAP data can be directly turned into music with minimal transformation. This is because PMAP data *is* music and computations done with PMAP data are computations done with music. Why is this important? Because PMAP is constructed so that the emotion which a PMAP data stream represents in the computation engine, will be similar to the emotion that a person "listening" to PMAP-equivalent melody would be. So PMAP can be used to calculate "feelings" and the result data will "sound like" the feelings calculated. This is will be clarified over the course of this paper.

In the previous subsection it was described how this paper is motivated by similarities between MIDI-type structures and the pulsed-processing [7] computation found in artificial and biological systems. It is further motivated by three other key elements which will now be examined: (i) the increasing prevalence of the simulation and communication of affective states by artificial and human agents/nodes; (ii) the view of music as the "language of emotions"; (iii) the concept of audio-display of non-audio data.

1.2 Affective Processing and Communication

It has been shown that affective states (emotions) play a vital role in human cognitive processing and expression [8]:

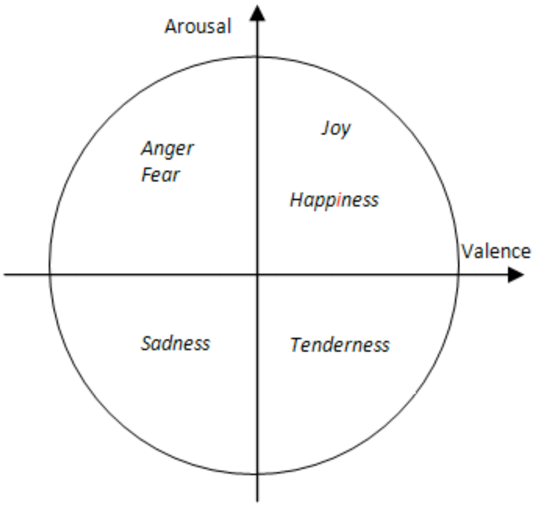
1. Universal and Enhanced Communication – two people who speak different languages are still able to communicate basic states such as happy, sad, angry, and fearful.
2. Internal Behavioral modification - a person’s internal emotional state will affect the planning paths they take. For example affectivity can reduce the number of possible strategies in certain situations – if there is a snake in the grass, fear will cause you to only use navigation strategies that allow you to look down and walk quietly. Also pre- and de-emphasising certain responses: for example if a tiger is chasing you, fear will make you keep running and not get distracted by a beautiful sunset, or a pebble in your path.
3. Robust response – in extreme situations the affective reactions can bypass more complex cortical responses allowing for a quicker reaction, or allowing the person to respond to emergencies when not able to think clearly – for example very tired, or in severe pain, and so forth.

As a result, affective state processing has been incorporated into robotics and multi-agent systems [9]. A further reason in relation to point (1) above and Human-Computer Interaction studies is that emotion may help machines to interact with and model humans more seamlessly and accurately [10]. So representation of simulation affective states is an active area of research.

The dimensional approach to specifying emotional state is one common approach. It utilizes an n-dimensional space made up of emotion “factors”. Any emotion can be plotted as some combination of these factors. For example, in many emotional music systems [11] two dimensions are used: Valence and Arousal. In that model, emotions are plotted on a graph (see Figure 1) with the first dimension being how positive or negative the emotion is (Valence), and the second dimension being how intense the physical arousal of the emotion is

(Arousal). For example "Happy" is high valence, high arousal affective state, and "Stressed" is low valence high arousal state.

Figure 1: The Valence/Arousal Model of Emotion



1.3 Music and Emotion

There have been a number of questionnaire studies done which support the argument that music communicates emotions [12]. Previous research [13] has suggested that a main indicator of valence is musical key mode. A major key mode implies higher valence, minor key mode implies lower valence. For example the overture of The Marriage of Figaro opera by Mozart is in a major key; whereas Beethoven's melancholic "Moonlight" Sonata movement is in a minor key. It has also been shown that tempo is a prime indicator of arousal, with high tempo indicating higher arousal, and low tempo - low arousal. For example: compare Mozart's fast overture above with Debussy's major key but low tempo opening to “Girl with the Flaxen Hair”. The Debussy piano-piece opening has a relaxed feel – i.e. a low arousal despite a high valence.

1.4 Sonification

Sonification [14] involves representing non-musical data in audio form to aid its understanding. Common forms of sonification include Geiger Counters and Heart Rate monitors. Sonification research has included tools for using music to debug programs [15], sonify activity in computer networks [16] and to give insight into stock market movements [17]. In the past, sonification has been used as an extra module attached to the output of the system under question.

A key aim of PMAP is to allow sonification in affective systems at any point in the processing path within the system. For example between two neurons in an artificial neural network, or between two agents in a multi-agent system, or between two processing modules within a single agent. The aim is to give the engineer or user quicker and more intuitive insight into what is occurring within the communication or processing path in simulated emotion systems by actually using simple music itself for processing and communication.

There are already systems which can take the underlying binary data and protocols in a network and map them onto musical features [18]. However PMAP is the only data processing model currently which is its own sonification and requires no significant mapping for sonifying. This is because PMAP data is limited to use in affective communications and processing, where music can be both data and sonification simultaneously. PMAP is not a new sonification algorithm, it is a new data representation and processing approach which is already in a sonified form.

This means that no conversion is needed between the actually processing / communication stream and the listening user - except perhaps downsampling. It also allows the utilization of such musical features as harmony and timing synchronization to be incorporated into the monitoring, when multiple modules / agents are being monitored simultaneously.

2 PMAP REPRESENTATION OF AFFECTIVE STATE

In the original monophonic PMAP [19] the data stream representing affective state is a stream of pulses. The pulses are transmitted at a variable rate. This can be compared to the variable rate of pulses in biological neural networks in the brain, with such pulse rates being considered as encoding information. In PMAP this pulse rate specifically encodes a represention of the arousal of an affective state. A higher pulse rate is essentially a series of events at a high tempo (hence high arousal); whereas a lower pulse rate is a series of events at a low tempo (hence low arousal).

Additionally, the PMAP pulses can have variable heights with 10 possible levels. For example 10 different voltage levels for a low level stream, or 10 different integer values for a stream embedded in some sort of data structure. The purpose of pulse height is to represent the valence of an affective state, as follows. Each level represents one of the musical notes C,D,Eb,E,F,G,Ab,A,Bb,B. For example 1mV could be C, 2mV be D, 3mV be Eb, etc. We will simply use integers here to represent the notes (i.e. 1 for C, 2 for D, 3 for Eb, etc). These note values are designed to represent a valence (positivity or negativity of emotion). This is because, in the key of C, pulse streams made up of only the notes C,D,E,F,G,A,B are the notes of the key C major, and so will be heard as having a major key mode – i.e. positive valence. Whereas streams made up of C,D,Eb,F,G,Ab,Bb are the notes of the key C minor, and so will be heard as having a minor key mode – i.e. negative valence.

For example a PMAP stream of say [C,C,Eb,F,D,Eb,F,G,Ab,C] (i.e. [1,1,3,5,3,4,5,6,7]) would be principally negative valence because it is mainly minor key mode. Whereas

[C,C,E,F,D,E,F,G,A,C] (i.e. [1,1,4,5,2,4,5,6,8]) would be seen as principally positive valence. And the arousal of the pulse stream would be encoded in the rate at which the pulses were transmitted. So if [1,1,3,5,3,4,5,6,7] was transmitted at a high rate, it would be high arousal and high valence – i.e. a stream representing ‘happy’ (see Figure 1). Where as if [1,1,4,5,2,4,5,6,8] was transmitted at a low pulse rate then it will be low arousal and low valence – i.e. a stream representing ‘sad’.

Note that [1,1,3,5,3,4,5,6,7] and [3,1,3,5,1,7,6,4,5] both represent high valence (i.e. are both major key melodies in C). This ambiguity has a potential extra use. If there are two modules or elements both with the same affective state, the different note groups which make up that state representation can be unique to the object generating them. This allows other objects, and human listeners, to identify where the affective data is coming from.

In non-simulated systems the PMAP data would be a stream of pulses. In fact in the first example below, a pulse-based data stream (MIDI) is used directly. However in performing the analysis on PMAP for simulation, it is convenient to utilize a parametric form to represent the data stream form. The parametric form represents a stream by a Tempo-value variable and a Key-mode-value variable. The Tempo-value is a real number varying between 0 (minimum pulse rate) and 1 (maximum pulse rate). The Key-mode-value is an integer varying between -3 (maximally minor) and 3 (maximally major).

3 PREVIOUS PMAP RESULTS

Monophonic PMAP has been applied and tested in a number of simulations. As there is no room here to go into detail, these systems and their results will be briefly described. They are:

a. A security team multi-robot system [19, 20, 21]
b. A musical neural network to detect textual emotion [19, 20, 21]

c. A stock market algorithmic trading and analysis approach [19, 22]
d. A multi-layered affective protocol for robust communication [20]

The security robot team simulation involved robots with two levels of intelligence: a higher level more advanced cognitive function and a lower level basic affective functionality. The lower level functionality could take over if the higher level ceased to work. A new type of logic gate was designed to use to build the lower level: musical logic gates. PMAP equivalents of AND, OR and NOT were defined, inspired by Fuzzy Logic. Brief descriptions of these are given later in this paper. It was shown that using a circuit of such gates, PMAP could provide basic fuzzy search and destroy functionality for an affective robot team. It was also found that the state of a three robot team was human audible by tapping in to parts of the PMAP processing stream.

As well as designing musical logic gates, a form of musical artificial neuron was defined. A simple two layer PMAP neural network was implemented using the MATLAB MIDI toolbox. The network was trained by gradient descent to recognise when a piece of text was happy and when it was sad. The tune output by the network exhibited a tendency towards "sad" music features for sad text, and "happy" music features for happy text. The stock market algorithmic trading and analysis system involved defining a generative affective melody for a stock market based on its trading imbalance and trading rate. This affective melody was then used as input for a PMAP algorithmic trading system. The simple system was shown to make better profits than random in a simulated stock market.

The multi-layered affective protocol utilized ideas from human expressive performance of music which supports that there are two levels of communication in a human's musical performance, both of which can separately communicate emotion. They are the compositional level and the expressive performance level. By applying computer expressive performance simulations to a PMAP data stream, it was shown that the affective state of the data stream could be communicated between computer nodes even when the underlying PMAP data was greatly distorted by simulated communications errors. A key element of this approach to PMAP comms robustness was that the expressive performance overlay would "sound like" its affective content when listened to by a person, in the same way that the underlying PMAP data would "sound like" the affectivity it contained.

4 PMAPh: PROPOSED PMAP HARMONIC EXTENSIONS

An addition is now proposed to this previous work – the utilization of harmony in PMAP. Before doing this, the basics of PMAP logic and connectionism will be re-iterated from previous papers.

4.1 Music Gates

Three possible PMAP gates were examined in [19] based on AND, OR and NOT logic gates. The PMAP versions of these are respectively: MAND, MOR and MNOT (pronounced "emm-not"), MAND, and MOR. So for a given stream, the PMAP-value can be written as $m_i = [k_i, t_i]$ with key-value k_i and tempo-value t_i. The definitions of the musical gates are (for two streams m_1 and m_2):

$$\text{MNOT}(m) = [-k, 1-t] \qquad (1)$$

$$m1 \text{ MAND } m2 = [minimum(k1,k2), minimum(t1,t2)] \qquad (2)$$

$$m1 \text{ MOR } m2 = [maximum(k1,k2), maximum(t1,t2)] \qquad (3)$$

These use a similar approach to Fuzzy Logic [23]. MNOT is the simplest – it simply inverts the key mode and tempo – minor becomes major and fast becomes slow, and vice versa. The best way to get some insight into what the affective function of the music gates is it to utilize music “truth tables”, which will be called Affect Tables here. In these, four representative state-labels – based on the PMAP-value system - are used to represent the four quadrants of the PMAP-value table: “Sad” for [-3,0], “Stressed” for [-3,1], “Relaxed” for [3,0], and “Happy” for [3,1]. Table 1 shows the music tables for MAND and MNOT. The columns marked KT-Value refer to the PMAP-value system.

Table 1: Music Tables for MAND and MNOT

MAND						**MNOT**			
State Label 1	*State Label 2*	*KT-value 1*	*KT-value 2*	*MAND value*	*State Label*	*State Label*	*KT-value*	*MNOT value*	*State Label*
Sad	Sad	-3,0	-3,0	-3,0	Sad	Sad	-3,0	3,1	Happy
Sad	Stressed	-3,0	-3,1	-3,0	Sad	Stressed	-3,1	3,0	Relaxed
Sad	Relaxed	-3,0	3,0	-3,0	Sad	Relaxed	3,0	-3,1	Stressed
Sad	Happy	-3,0	3,1	-3,0	Sad	Happy	3,1	-3,0	Sad
Stressed	Stressed	-3,1	-3,1	-3,1	Stressed				
Stressed	Relaxed	-3,1	3,0	-3,0	Sad				
Stressed	Happy	-3,1	3,1	-3,1	Stressed				
Relaxed	Relaxed	3,0	3,0	3,0	Relaxed				
Relaxed	Happy	3,0	3,1	3,0	Relaxed				
Happy	Happy	3,1	3,1	3,1	Happy				

Taking the MAND of two melodies, the low tempos and minor keys will dominate the output. Taking the MOR of two melodies, then the high tempos and major keys will dominate the output. Another perspective: the MAND of the melodies from Moonlight Sonata (minor key) and the Marriage of Figaro Overture (major key), the result would be mainly influenced by Moonlight Sonata. However if they are MOR’d, then the Marriage of Figaro Overture key mode would dominate. The MNOT of Marriage of Figaro Overture would be a minor key version. The MNOT of Moonlight Sonata would be a faster major key version. It is also possible to construct more complex music functions. For example MXOR (pronounced "mex-or"):

$$m_1 \text{ MXOR } m_2 = (m_1 \text{ MAND MNOT}(m_2)) \text{ MOR } (\text{MNOT}(m_1) \text{ MAND } m_2) \qquad (4)$$

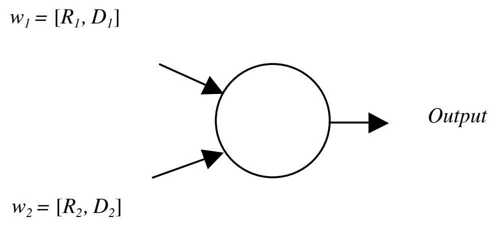
4.2 Musical Neural Networks

The use of timed pulses in Spiking Neural Networks supports an investigation into using PMAP pulses in Artificial Neural Networks. In particular in a neural network application in which emotion and rhythm are core elements. A form of learning artificial neural network which uses PMAP is now given. These artificial networks take as input, and use as their processing data, pulsed melodies. A musical neuron (muron – pronounced MEW-RON) is shown in Figure 2.

The muron in this example has two inputs, though a muron can have more than this. Each input is a PMAP melody, and the output is a PMAP melody. The weights on the input w_1 and w_2 are each two element vectors which define a key mode transposition, and a tempo change. A positive R_k will make the input tune more major, and a negative one will make it more minor. Similarly a positive D_t will increase the tempo of the tune, and a negative D_t will reduce the tempo. The muron combines input tunes by superposing the spikes in time – i.e. overlaying them. Any notes which occur at the same time are combined into a single note with the highest pitch being retained. This retaining rule is fairly arbitrary but some form of non-random decision should be made in this scenario (future work will examine if the "high retain" rule adds any significant bias). Murons can be combined into networks, called musical neural networks, abbreviated to "MNN". The learning of a muron involves setting the weights to give the desired output tunes for the given input tunes. Applications for which

PMAP is most efficiently used are those that naturally utilize temporal or affective data (or for which internal and external sonification is particularly important).

Figure 2: A Muron with two inputs.



4.3 Harmonic Extensions for PMAP

One key element of music which is not shared necessarily by other areas such as visual display, is the ability for two musical streams to combine in a single stream which has its own gestalt meaning: i.e. a harmony. This allows the potential for auditory display of multiple streams of data. For example a major PMAP stream at a medium tempo would tend to communicate happiness, whereas a minor PMAP stream at a medium tempo would tend to communicate sadness. If the two are synchronized and combined together as a harmony this will probably cause it to sound quite unpleasant, since 4 of the pitches in a major key clash harmonically with 3 of the pitches in a minor key. This harmonic "clashing", known as dissonance – indicates that the two streams have very different keys and thus - by the original PMAP definition - very different valences.

In Monophonic PMAP the harmonic effects of multiple streams are only detectable by humans who listen to the streams simultaneously. This property was discussed in [20] as being useful for a human commander tapping into the processing of the robot team. An extension is now proposed to PMAP, called Harmonic PMAP, or PMAPh (pronounced "peemaff"). In PMAPh the arousal of a data stream is still defined by its tempo. However PMAPh also allows for polyphonic data streams, with up to two simultaneous notes occurring at any one time in the data stream. In polyphonic PMAPh data streams the valence of a polyphonic data stream when there are simultaneous pitches is based on the consonance as calculated using Table 2. Consonance in PMAPh is equivalent to Key-value in monophonic PMAP. A low consonance means that the two simultaneous notes clash harmonically (are dissonant), and a high consonance means they sound pleasant together. In monophonic PMAP a high Key-value means more of the melody notes are in a major (more positive sounding) key, whereas a low Key-value means more of the notes are in a minor (less positive sounding) key.

Table 2: Reference table for PMAPh valence

Note MIDI Difference in same octave	**Name**	**PMAPh Consonance**
0, 4, 7	unison / 3rd / 5th	3
5, 9	4th / 6th	1
2,3,8,10	2nd / minor 2nd / minor 6th / minor 7th	-1
1, 6, 11	minor 2nd / minor 5th / major 7th	-3

So the unison, 3rd and 5th intervals have the highest valence, and the semitone, tritone and major 7th intervals have the lowest valence. The values in the first two columns of this table come from a Genetic Algorithm harmony system developed for a system for emotional expression by algorithmic harmony composition [25]. This table allows the musical logic gates and musical neural networks defined in PMAP to be used in PMAPh with polyphonic inputs.

The implementation of a musical neuron in [19] and [21] was originally defined as selecting the highest of two pitches when two pitches occurred simultaneously in time. In

PMAPh it is defined as an interval made up of the highest and lowest simultaneous pitches. Once again this is fairly arbitrary it means that a unitary musical neuron (i.e. with unitary weights) can be used to combine two monophonic PMAPh streams into a polyphonic PMAPh stream.

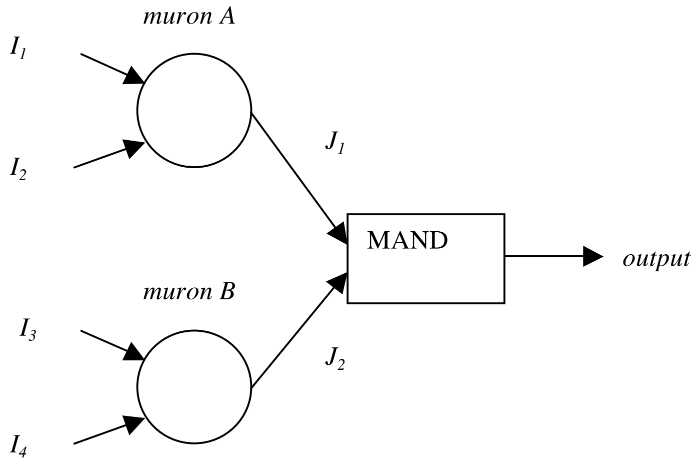
To examine the differences between PMAP and PMAPh consider the MAND music gate. In a PMAP MAND gate both input melodies have to be major key for the output to have a major key, otherwise the output has a minor key. The tempo output is the minimum tempo of the two inputs. It implies that both inputs have to have high valence for the output to have high valence, and both inputs have to have high arousal for the output to have high arousal. In PMAPh when the stream is not monophonic, its valence is positively correlated to the consonance defined in Table 2. Table 3 shows the functioning of a MAND gate in PMAPh for just the most consonant and least consonant intervals. (Note that because simple harmonies are more to do with pitch than with rhythm, the following discussion does not go into detail about tempo in PMAPh.)

Table 3: MAND table for Maximum Consonances

MAND (maxima only)					
Input Interval 1	*Input Interval 2*	*Consonance 1*	*Consonance 2*	*MAND value*	*Valence*
1, 6, 11	1, 6, 11	-3	-3	-3	Low
1, 6, 11	0, 4, 7	-3	3	-3	Low
0, 4, 7	1, 6, 11	3	-3	-3	Low
0, 4, 7	0, 4, 7	3	3	3	High

Figure 3 gives an example application that differentiates PMAPh from monophonic PMAP. The circles in Figure 3 are unitary musical neurons (i.e. with weights that do not change the input tunes features). Suppose the 4 inputs are monophonic and *time-quantized*. The time quantized element is important here, as it increases the likelihood that two pitches in two separate melodies could occur at the same time, thus creating harmonies. Suppose first that the tunes on I_1, I_2, I_3 and I_4 are melodies which have no notes that coincide in time. Then the output of the circuit in Figure 3 will be the same for monophonic PMAP and for PMAPh by the definition of the musical neuron discussed above. However the more notes that coincide in time, the more different the outputs for PMAP and PMAPh will be.

Figure 3: A PMAPh demonstration circuit



Suppose that all notes coincide in time. In monophonic PMAP the outputs of murons A and B (J_1 and J_2) will work as "pitch gates", only allowing through the highest of the two pitches, without changing the timing. Then the MAND gate will work as described in previous PMAP work [19] and Table 3 – i.e. if the keymodes of J_1 and J_2 are both major the output key will be major, but if one or more is minor, the output will have a minor key mode. In PMAPh however the murons will output harmonies to J_1 and J_2, where J_1 contains the intervals made up by simultaneous notes in I_1 and I_2; and J_2 made up of I_3 and I_4 notes. In PMAPh the tempo behavior is the same as in PMAP, and since all notes in I_1 and I_2 (and I_3

and I_4) are simultaneous, they must have the same tempo. Hence the final output of the circuit in Figure 3 will have the same tempo as the initial inputs.

However the valence / consonance behavior of a MAND gate in PMAPh is defined using Tables 2 and 3, in a way analogous to the MAND gate in monophonic PMAP is for Key-mode. In PMAPh the MAND gate gives a very consonant ("pleasant sounding") output harmony when the two input harmonies are consonant, and a dissonant ("clashing") output when the two input harmonies are dissonant.

So what is the overall effect of this circuit when the four input melodies are synchronized? If the music table is calculated for Figure 3, it is seen that the output is consonant (pleasant sounding) when the inputs are all the same key-mode, or are pair-wise the same by muron. Affectively this means that the output will be high valence when all the inputs have the same valence, or inputs for both murons have individually the same valence. To describe this more iteratively, suppose all inputs initially have the same valence – the output valence will be high. Now suppose one input deviates from the others, then the output valence will go low. However suppose an input on the same muron as the deviating muron also deviates from the original valence – then the output will go high again.

This is a very different functionality to the monophonic PMAP version and allows for selective detection of non-pairwise valence deviations. If each input came from a robot and the robots were paired into two teams, then the system would help to monitor that both robot teams were matched in how positive or negative they were "feeling". The more their valences deviated pair-wise – the lower the output valence would become. For the security robots application [19] this could be used to detect if paired robots were moving too far apart – since being close together would increase the likelihood that they were experiencing the same environment and thus had similar valences. As with monophonic PMAP this state detection output can be used for further calculations, but furthermore if it is probed and heard by a human being, it will be aurally significant: the more the paired robots deviate from each other, the more “clashing” the PMAPh stream harmonies sound to a human ear. Thus the PMAPh stream can be used for audio display as well as computation.

To demonstrate this in action, a new simulation from the multi-robot system in [19] was run. Note that the PMAPh approach is not being used here for the robots to communicate with each other. It is being used to allow each individual robot to process affective information internally. The PMAP equations used in [19] are shown in equations 5 and 6, but will not be explained in detail here.

$$\text{WEAPON} = \text{DetectOther MAND MNOT(FriendFlag)} \tag{5}$$

$$\text{MOTOR} = \text{WEAPON MOR MNOT(DetectOther)} \tag{6}$$

The short melodies which robots use as the building blocks for their PMAP calculations are called “identives”. The identives used in [19] have been updated here so as to maximize the harmonic effects. This is because identives only interact harmonically in interesting ways if the “key” notes are in similar positions. Thus is two identives are such that the main notes identifying the differences between C major and C minor never coincide in time, then there will rarely be significant dissonance (or consonance).

The first 500 notes of the WEAPON object are shown in Figure 4 at the end of this paper. Tempo is fixed in this example so as to emphasize the harmonic effects. Security robots 1, 2, 3, and 4 are shown respectively in the figure rows. The simulation is run so that Robots 1 and 2 start off close to each other in the environment, as do robots 3 and 4. Figure 5 shows the consonance of the outputs of the two neurons and the resulting MAND output. In Figure 5, three sections of the MAND output have been marked up. These are the areas with the lowest valence (i.e. highest dissonance). Anyone tapping in to this PMAP stream and listening to it will hear a more dissonant tune. The three sections of the Robot's PMAP WEAPON stream which cause these periods have been marked up as well in Figure 4. A major key stream in WEAPON causes the robot's weapon to fire, whereas a minor key stream means the weapon is not firing. It can be seen that during the first marked-up period in Figure 4, Robot 4 is firing and Robot 3 is not. During the second period Robot 1 is firing and Robot 2 is not. During the third period Robot 4 is firing and Robot 3 is not. Thus if the robots are considered as being paired off into teams of 1 and 2, and 3 and 4, then the output of the PMAPh circuit in Figure 3 becomes dissonant when one or more of the two robot teams are mismatched in their weapon behaviour, as discussed earlier.

5 CONCLUSIONS AND FUTURE WORK

This paper has given an overview of the concept of Pulsed Melodic Affective Processing, a complementary approach in which computational efficiency and power are more balanced with understandability to humans; particularly where computation addresses rhythmic and simulated emotion processing. Previous work on PMAP has demonstrated the utilization of musical logic gates, and musical artificial neurons.

A key contribution of PMAP is at the research interface between non-standard computation and human-computer interaction. In normal circuit and network sonification, a probe needs to be placed at the node we desire to sonify, and that data then needs to be fed into a sonification algorithm to be converted into meaningful sounds for the user. However if, in the case of affective circuits and networks, the underlying data uses the PMAP representation, then no sonification algorithm is needed. The data is already in the form of a melody which represents the affective state of the data – in other words the data representation is its own sonification. There are systems which allow the sonification of network data through separate data sonification algorithms. These systems will take the underlying binary data and protocols, map them onto features, and then play these features. However PMAP is the only data processing and transmission model currently which is its own sonification and requires no significant mapping. This is because PMAP is limited to use in affective communications and processing, and such affective states can be represented and computed in many cases by musical data anyway.

Because of music's special ability to turn two separate streams into a single harmonic gestalt with its own affective implications, it is argued that extensions of PMAP into polyphony should be investigated. An extension of PMAP to PMAPh has been proposed, together with an example that demonstrated some significant functionality differences.

There are a significant number of issues to be further addressed with PMAP in general. For example the actual application of the music gates discussed depends on the level at which they are to be utilized. The underlying data of PMAP (putting aside for a moment the PMAP-value representation used for simplicity earlier) is a stream of pulses of different heights and pulse rates. At the digital circuit level this can be compared to VLSI hardware spiking neural network systems [26] or VLSI pulse computation systems. As has been mentioned, a key difference is that the pulse height varies in PMAP, and that specific pulse heights must be distinguished for computation to be done. But assuming this can be achieved, then the gates would be feasible in hardware. It is probable that each music gate would need to be constructed from multiple VLSI elements due to the detection and comparison of pulse heights necessary.

The other way of applying at a low level, but not in hardware, would be through the use of a virtual/simulated machine. So the underlying hardware would use standard logic gates or perhaps standard spiking neurons. The idea of a virtual/simulated machine may at first seem contradictory, but one only needs to think back twenty years when the idea of the Java Virtual Machine would have been unfeasible given current processing speeds then. In 5-10 years current hardware speeds may be achievable by emulation; and should PMAP-type approaches prove useful enough, would provide one possible implementation.

Other issues to be further examined include: is the rebalance between efficiency and understanding useful and practical, and also just how practical is sonification - can sonification more advanced than Geiger counters, heart rate monitors, etc. really be useful and adopted? The valence/arousal coding provides simplicity, but is it sufficiently expressive while remaining simple? Similarly it needs to be considered if a different representation than tempo/key mode would be better for processing or transparency. Furthermore PMAP also has a close relationship to Fuzzy Logic and Spiking Neural Networks – so perhaps it can adapted based on lessons learned in these disciplines.

References

1. Miranda, E., Adamatzky, A. and Jones, J. (2011) Sounds Synthesis with Slime Mould of Physarum Polycephalum, *Journal of Bionic Engineering* 8, pp. 107-113 Elsevier
2. Miranda, E. R., Bull, L., Gueguen, F., Uroukov, I. S. (2009). Computer Music Meets Unconventional Computing: Towards Sound Synthesis with In Vitro Neuronal Networks, *Computer Music Journal*, Vol. 33, No. 1, pp- 9-18.
3. Juslin, P. and Sloboda, J. (2001) Music and Emotion: Theory and Research. Oxford University Press, Oxford, UK.
4. Bohte, S., Kok, J., La Poutre, H. (1997) Spike-prop: Error-backpropagation in multi-layer networks of spiking neurons. *Neurocomputing*, 48(1-4), 17-37.
5. Perrinet, L., Samuelides, M., Thorpe, S. (2004) Coding static natural images using spiking event times: do neurons cooperate? *IEEE Trans Neural Networks* 15 1164-75.

6. Chen N, Yu J, Qian H, Ge R, Wang J-H. (2010) Axons Amplify Somatic Incomplete Spikes into Uniform Amplitudes in Mouse Cortical Pyramidal Neurons. *PLoS ONE* 5(7): e11868.
7. Miller, J., Young, W. (1996) Simple Pulse Asynchronous State Machines, *Proceedings of International Symposium on Circuits and Systems*, IEEE
8. Malatesa, L., Karpouzis, K., Raouzaiou, A. (2009) Affective intelligence: the human face of AI, In *Artificial intelligence*, Springer-Verlag.
9. Banik, S., Watanabe, K., Habib, M., Izumi, K. (2008) Affection Based Multi-robot Team Work, In *Lecture Notes in Electrical Engineering*, pp. 355--375.
10. Picard, R. (2003) Affective Computing: Challenges, *International Journal of Human-Computer Studies*, Vol. 59, No. 1-2, pp. 55--64.
11. Kirke, A., Miranda, E. (2011) Emergent construction of melodic pitch and hierarchy through agents communicating emotion without melodic intelligence, In *Proceedings of 2011 International Computer Music Conference* (ICMC 2011), International Computer Music Association.
12. Juslin, P., Laukka, P. (2004) Expression, perception, and induction of musical emotion: a review and a questionnaire study of everyday listening. *Journal of New Music Research*, vol. 33, pp. 216--237.
13. Juslin, P. (2005) From Mimesis to Catharsis: expression, perception and induction of emotion in music, In *Music Communication*, pp. 85--116, Oxford University Press.
14. Cohen, J. (1994) Monitoring Background Activities, In *Auditory Display: Sonification, Audification, and Auditory Interfaces*, pp. 499--531.
15. Vickers, P., Alty, J. (2003) Siren songs and swan songs debugging with music. *Communications of the ACM*, Vol. 46, No. 7, pp. 87--92.

16. Gilfix, M., Couch, A.L. (2000) Peep (the network auralizer): Monitoring your network with sound. *Proceedings of the 14th System Administration Conference* (LISA 2000), 109–117, New Orleans, Louisiana, USA, 3–8 December 2000. The USENIX Association.
17. Kirke, A., Miranda, E. (2012) Application of Pulsed Melodic Affective Processing to Stock Market Algorithmic Trading and Analysis, *Proceedings of 9th International Symposium on Computer Music Modeling and Retrieval* (CMMR2012), London
18. Chafe, C., Leistikow, L. (2001) Levels of Temporal Resolution in Sonification of Network Performance, *Proceedings of 2001 International Conference on Auditory Display*, Finland.
19. Kirke, A., Miranda, E. (In Press) "Pulsed Melodic Processing – the Use of Melodies in Affective Computations for Increased Processing Transparency". *Music and Human-Computer Interaction*, S. Holland, K. Wilkie, P. Mulholland and A. Seago (Eds.), London: Springer.
20. Kirke, A., Miranda, E. (Submitted) Pulsed Melodic Affective Processing – Musical Structures for Increasing Transparency in Emotional Computation, *Transactions of the Society for Modelling and Simulation International*, SAGE
21. Kirke, A. and Miranda, E. (2012). "Pulsed Melodic Affective Processing – Using Music for Natural Affective Computation and Increased Processing Transparency", *Neural, Parallel, and Scientific Computations* 20:227–240, Dynamic Publishers, USA.
22. Kirke, A. and Miranda, E. R. (2012). Application of Pulsed Melodic Affective Processing to Stock Market Algorithmic Trading and Analysis. *Proceedings of 9th International Symposium on Computer Music Modeling and Retrieval* (CMMR2012), London (UK).
23. Marinos, P. (1969) Fuzzy Logic and Its Application to Switching Systems, *IEEE Transactions on Computers C*, Vol. 18, No. 4, pp. 343--348.
24. Haykin, S.,: Neural Networks a Comprehensive Foundation, Prentice Hall (1994).

25. Kirke, A, Davies, G. (2013) Open Outcry. *Invited Talk, The Royal Institution*, London, UK

26. Indiveri, G.; Chicca, E.; Douglas, R.: A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity, *IEEE Transactions on Neural Networks*, Vol. 17, No. 1, pp. 211--221 (2006).

THE STORY OF TELEBRAIN:
A multi-performer telematic platform for performatization

Abbreviated Title: The Story of Telebrain

Kristin Grace Erickson

Performer-Composer D.M.A. Program
The Herb Alpert School of Music
California Institute of the Arts
24700 McBean Pkwy
Valencia, CA 91355
United States of America
kerickson@calarts.edu
+1 (850) 656-8879

ABSTRACT

This paper presents Telebrain, a browser-based *performatization* platform invented for organizing real-time telematic performances.

Performatization is the human performance of algorithms. When computers and humans performatize cooperatively, the human-computer interaction (HCI) becomes the location of computation. Novel modes of machine-human communication are necessary for organizing performatizations. Telebrain is designed to facilitate machine-human languages.

Capitalizing on the ubiquity and cross-platform compatibility of the Internet, Telebrain is an open-source web application supporting PerPL (Performer Programming Language), a human-interpreted configurable language of multi-media instructions used to program performers.

Telebrain facilitates a variety of performance disciplines such as music, theater, dance, computational performance, networked scoring (image and audio), prompted improvisation, real-space multi-player gaming, collaborative transdisciplinary karaoke and quantum square-dancing. (http://telebrain.org)

Keywords

Telebrain, PerPL, telematic, performatization, improvisation, music, unconventional computing, multi-player, multi-performer, M.T.Brain, interdisciplinary, transdisciplinary, human-interpreted programming language, audio score, performance game

INTRODUCTION

Visualization, sonification, and auralization are established practices for representing data and computational outputs. This paper develops *performatization* as the practice of performing data, and of performing computation on data, through music, theatre, dance, improvisation, and other transdisciplinary frameworks. A multi-performer, real-time, browser-based telematic platform for distributing a large volume of performatization cues is introduced for creating large scale human-performed computations through the arts. This telematic performatization platform is called Telebrain.

My recent work has involved the iterative design of performatization systems, entangling research and development with a collaborative artistic practice. This has resulted in a non-linear history of complex intersections between performance and computation. Heuristic methods of performatization originated from the analysis of improvised music and theater games. Both used rules to organize improvisations facilitating simple translations between performance instructions and computer algorithms. Creative experiments distributing performance instructions culminated with the invention of Telebrain.

Telebrain is a web-based performatization platform for collectively developing and distributing PerPL (Performer Programming Language), pronounced *purple.* A multi-media, multimodal programming language for machine-human communication, PerPL integrates language, logic, and art into a single symbolic instructional paradigm. Allowing evolutionary specification, PerPL relies on syntax and context plasticity provided by human interpretation, adapting its rules through implementation.

PerPL programs are typically distributed by a computer to parallel human brains. PerPL executes traditional SIMD (Single Instruction, Multiple Data) and MIMD (Multiple Intruction, Multiple Data) processing when multiple performers interpret the instructions — instructions are interpreted by *vectors* of performers.

By expanding computational architecture to include performance, the human-computer interaction (HCI) becomes a location for computation. As PerPL programs are interpreted by performers, computation emerges as artistic expression. Revolutionalized modes of communication, between creator, process, output, and observer, unleashes novel paradigms of creative potential yet to be fully conceived.

Although any type of performance can performatize, music functions doubly as performatization and as a mode of instruction. Music's ability to signify, encapsulate, sequence, and coordinate simultaneity contributes to a computational paradigm where perceivable machine transmissions are required.

In addition to serving as the launching point for conceiving performatization, sonification and auralization provide a scientific basis for applying the semiotic agency of music and sound to an audio-based human-interpreted programming language.

BACKGROUND

Sonification, Auralization and Performatization

Sonification is the mapping of data into the audio domain. For thousands of years sound has been used to represent information heard as striking clocks and bell towers. Pythagoras used measurements of the stars and planets to define a musical scale. The stethoscope, geiger counter, and telegraph are all sonification inventions.[1]

Since the birth of Information Theory, the capacity for sound to transmit information has been researched to greater depths. Perceptual thresholds have been tested to determine the amount and types of multivariate data discernible through sound.[2] Some sonification techniques encode data into the acoustic parameters of sound waves while other techniques map data into culturally-determined systems of musical organization.[3]

Auralizations, also called Auditory Displays, are a special type of sonification where sound represents the output of a computer process.[4] Considered the audio equivalent of a computer monitor, auralization is used to communicate hierarchical and navigational information to blind computer users.[5]

Auralization research has also focused on warning signals in airplane cockpits, since pilots are inundated with information and certain data needs to make itself known in high-stress conditions.[6] Different types of warning signals have been developed and tested for effectiveness in relation to the types of data needing to be communicated and the cognitive response to the sounds.[7] The *performance* of piloting a plane is aurally-informed by audio representations of the plane's internal systems in addition to the atmospheric systems through which the plane is flown.

Extending sonification and auralization, performatization is the human performance of algorithms. Unlike visual displays and auralizations, where pixels and sounds illustrate the output of a computer process, performatizations *are* the algorithmic process. Performatizations compute.

Performatizing Bubble Sort

The following description of a hypothetical performatization refers to a sorting algorithm found in most introductory algorithm textbooks.

The Bubble Sort algorithm begins by comparing the first two values in a list. If the values are out of order, Bubble Sort swaps the positions of the two values before comparing the next two values. When the end of the list is reached, Bubble Sort performs another iteration by returning to the beginning of the list and repeating the comparison process. If no further swapping is necessary at the end of an iteration, the sort is complete. Performatizing a Bubble Sort algorithm involves performing the individual comparisons and iterations of the algorithm.

To prepare, each performer wears a different number on their shirt. The performance begins by randomly shuffling a group of performers into a single line — into an array. An additional performer named *Deictor* functions as an array pointer and binary flag. *Deictor* begins each iteration by literally pointing to the first performer in the line. If the number worn by the performer to the left is less than the number worn by the performer, the two performers swap positions in line. *Deictor* steps through the list by pointing to the next performer in line and performing additional comparisons.

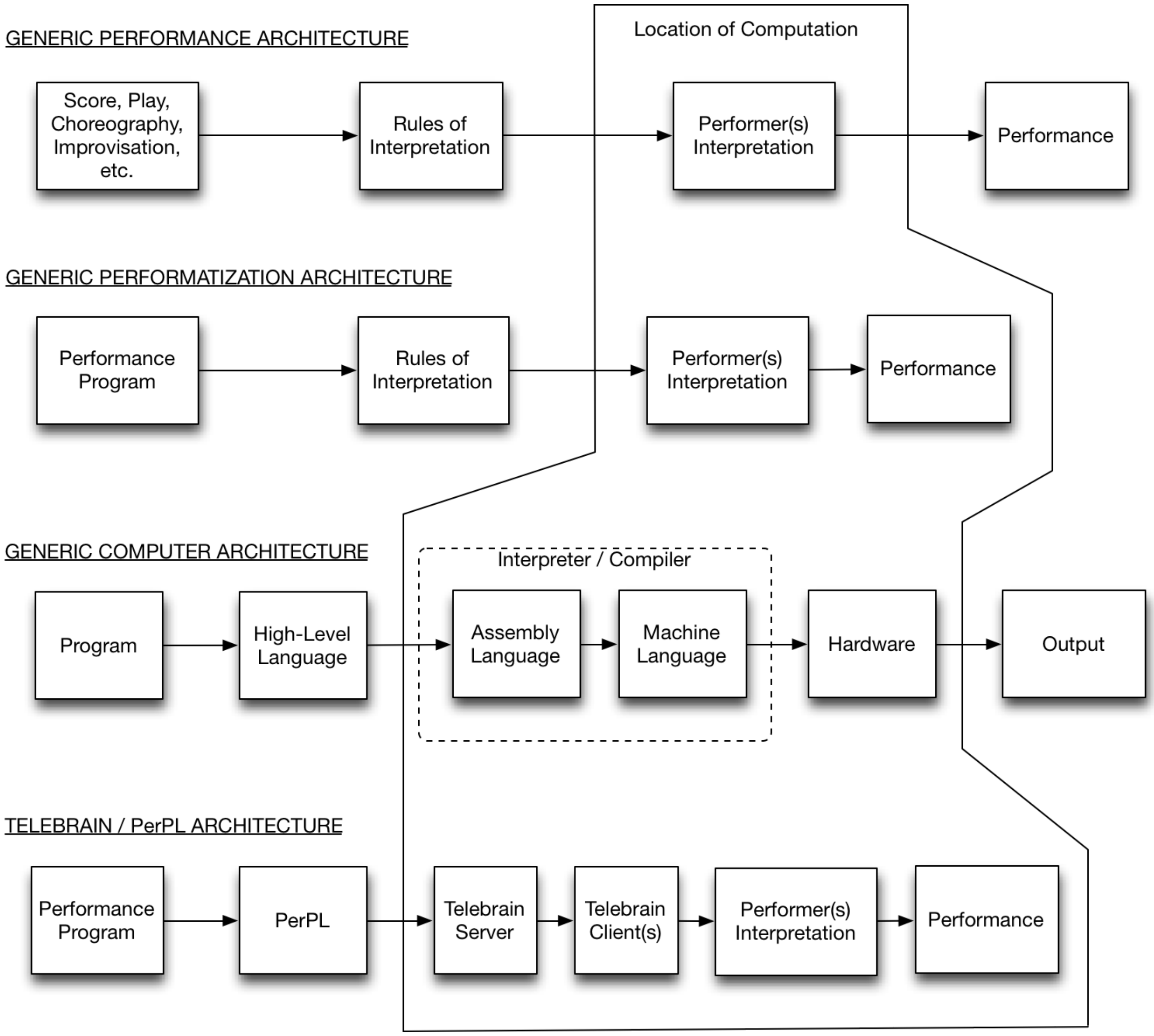
To indicate when the sort is complete, *Deictor* raises one arm at the beginning of each iteration, lowering the arm when a swap occurs. The raised arm functions as a binary flag signaling if additional iterations are required. If *Deictor's* arm is down at the end of an iteration, the list is not sorted and requires another iteration. If *Deictor's* arm is raised at the end of an iteration, the sort is complete and the performers are in numerical order from left to right.

This example can be extended to singing. Instead of numbers worn on clothing, each performer is assigned a unique pitch to sing. The performers sing their notes to compare pitches. If the pitches are out of order, the performers swap positions in line. Each iteration through the line produces a unique melodic contour and the sorting process is heard as melodic lines gradually developing toward an ascending pattern of notes.

Generally, computer science is concerned with algorithm efficiency, linking the qualitative value of an algorithm to the amount of resources a computational process requires. When performatizing, efficiency is overridden by the aesthetic quality and experiential value of the performance. In experimental performance, breaking rules reveals novel performance potential — malfunction leads to success, to change. Relieving algorithms of their need to achieve may expand the cultural applications of computation.

In Bubble Sort, the predetermined outcome is the *sorted* state of the list, and the algorithmic process produces this desired outcome. In traditional performance, the predetermined outcome is the performance itself. Performatizations regard performances as processual, and therefore a location where algorithms compute, as shown in Image 1.

Image 1. Architecture Comparisons



Removing the goal-state of the Bubble Sort performatization undermines the necessity for accurately evaluated comparisons. Now, an individual comparison can result in two players swapping positions <u>because they want to</u>. Performers have brains. Introducing human decision-making into an algorithmic process results in an algorithmic process capable of self-determining its outcome.

From an aesthetic standpoint, an ascending pattern of notes is not necessarily the preferred outcome of a Bubble Sort performatization. The experience of singing and listening to the melodic contours of previous iterations informs the swapping decisions of subsequent iterations, systematically democratizing aesthetic decisions of collaboration. Boolean probabilities correlate to applications of aesthetic criteria. Logic gates built from human personality and interaction self-organize patterns of melodic contour iterations, entangling computation, game theory, and aesthetics.

Rules of performatizations must be communicated to and understood by performers. Everyday language can explain simple algorithms, but performatizing complex processes requires a configurable language optimized for communicating real-time processes to multiple performers.

Cobra

> … the pieces slowly evolved into complex on-and-off systems … I eliminated the timeline. What remained were scores that did not refer to sound or time — two parameters traditionally inseparable from the art of music — but were a complex set of rules that, in a sense, turned players on and off like toggle switches to such a complicated degree that it didn't really matter what the content was.
>
> — John Zorn, "The Game Pieces"

John Zorn's improvised music game *Cobra* serves as a key model for *performing a system* using a configurable metalanguage to organize performances. The rules of *Cobra* enable self-organizing performance interactions to collectively embody a complex configurable system.

Cobra is played using color-coded cue cards inscribed with symbols. Players use hand signals to indicate desired cues to a prompter. The prompter presents the requested cue card to the ensemble before activating the cue by lowering the card. Cues are often called in quick succession causing sudden shifts in improvised material to create a fragmented musical experience. *Cobra* includes an embedded game allowing guerrilla factions to subvert the prompter/player relationship and subsequent music.

Cues initiate new musical information or modify existing musical information. Some cues require everyone to play while others require specific player assignments or sequence. Memory Cues associate played music with placeholders so previous musical states of the game can be recalled.

Cobra illustrates an efficient and practical communication system for organizing music improvisation, but not without limitations. Since *Cobra* relies on musical material improvised by experienced musicians, the cues designate imprecise qualities of music bypassing deeper levels of specificity. A performatization language designed to maximize its potential applications necessitates a capacity to communicate between extremities of precision. Since cue cards require continual eye contact between player and prompter, the system is impractical for performances involving body movement. To resolve the performance constraints of *Cobra's* visual cueing system, audio cueing systems were investigated for communicating further gradations of instruction specificity.

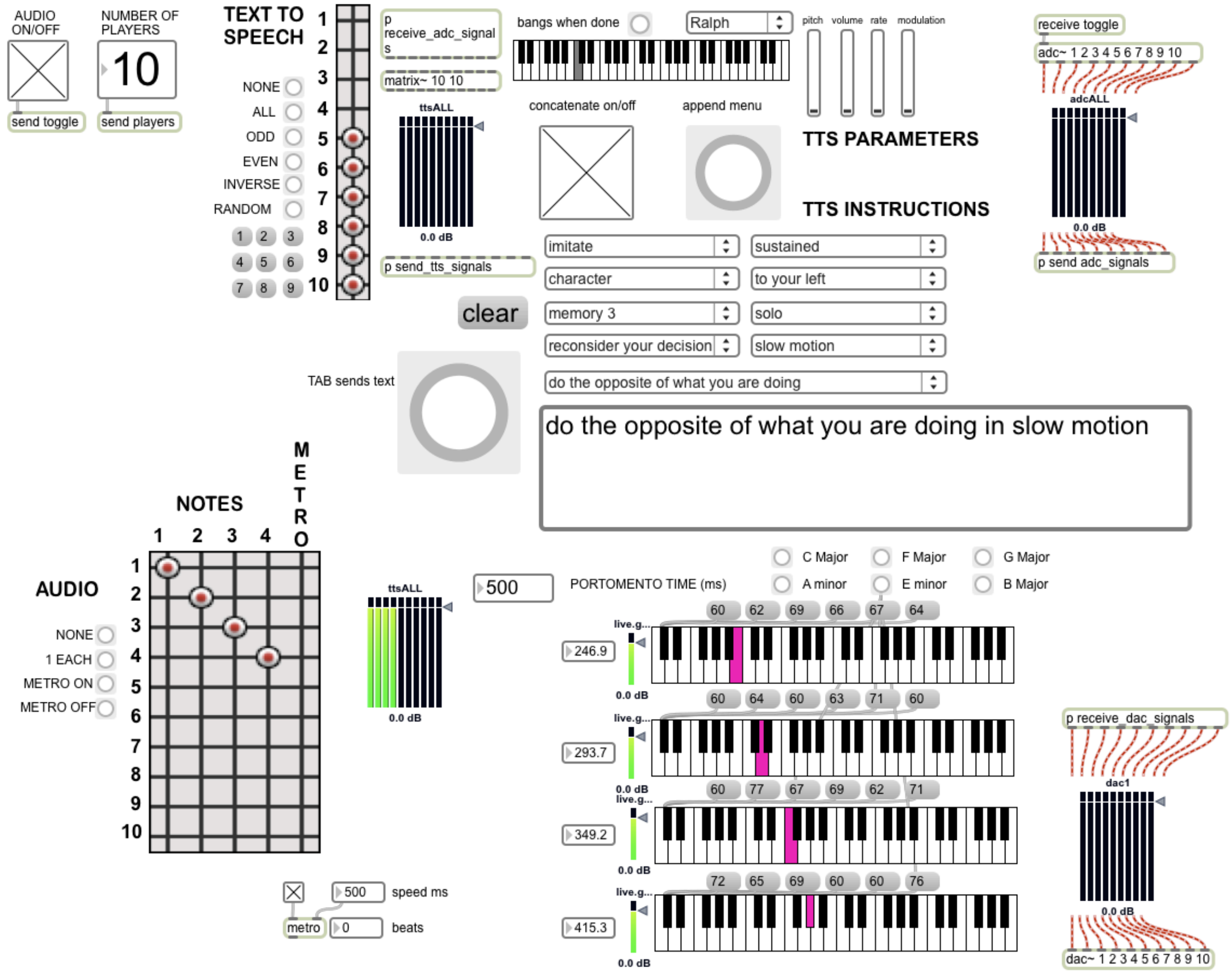
M.T.BRAIN

M.T.Brain (Music Theater Brain) is a Max/MSP patch developed for distributing real-time audio instructions to multiple performers. Using an audio routing matrix, parallel channels of audio are sent to performers through a ten-channel sound card, long cables, and headphones. Performers wear large numbers because performer interactions are indicated by channel number, as shown in Image 2.

Image 2. M.T.Brain Performance

A prompter operating M.T.Brain routes preset and real-time audio instructions to performers, as shown in Image 3. Typed instructions are spoken by an artificial text-to-speech voice. During a performance, new instructions can be saved for repeated use. A collection of easily-identifiable non-speech sounds are used to represent learned instructions or to indicate precisely when to perform an instruction. Like components of a configurable symbolic language, audio cues are concatenated with text-to-speech outputs to construct longer instructions, a prototype of PerPL.

Image 3. M.T.Brain Max/MSP Prompter Interface

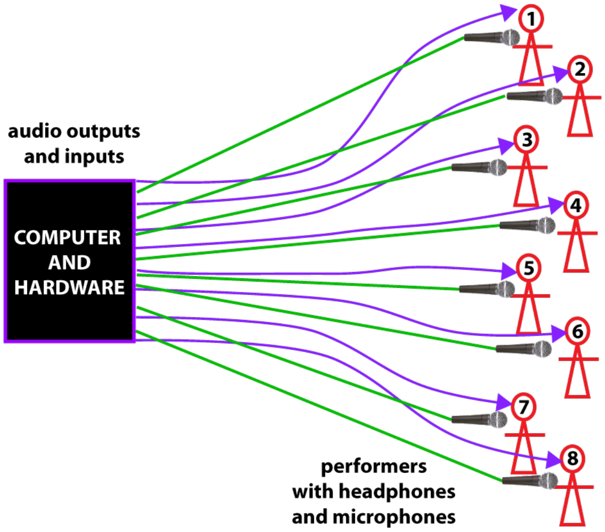


do the opposite of what you are doing in slow motion

Adjusting to the number of performers, M.T.Brain provides shortcuts for performer routing configurations including: all channels, one channel, odd or even channels, randomly selected groups of channels, and unselected complementary channels.

A metronome and four sine tones with frequency presets can be routed concurrently with audio instructions. Ten microphone inputs can be routed through M.T.Brain enabling real-time spoken instructions to be heard during a performance, as shown in Image 4.

Image 4. M.T.Brain Architecture



Preset audio cues inspired by improvised music and theater games can be used to organize live collaborative transdisciplinary improvisation. Styles of M.T.Brain performances are influenced by the prompter's approach to cueing and routing. Prompters can extend traditional modes of performance organization by configuring M.T.Brain to support the desired outcome.

For musical results, a prompter can take advantage of the temporal accuracy resulting from M.T.Brain's hard-wired parallel audio connections. By algorithmically scattering time-dependent instructions between multiple channels, so each performer receives 1/10th of the instructions constituting a complete gesture, the immediate link from computer-to-audio-to-ear-to-brain gives rise to tight rhythmic precision and multi-performer gestures to be heard — with little or no rehearsal. M.T.Brain performers can function like hammers on a player piano.

A prompter can provoke a spontaneous song and dance routine by routing lyrics to a lead vocalist who improvises the melody of a song. Simultaneously, the four sine tones are routed to the remaining performers who sing an accompaniment based on the four-voice chord presets the prompter triggers. M.T.Brain's parallel in-ear sine tones simplify singing dissonant and microtonal harmonies and changes. The sine tones are layered with a metronome and audio instructions for synchronizing choreography instructions distributed according to channel number.

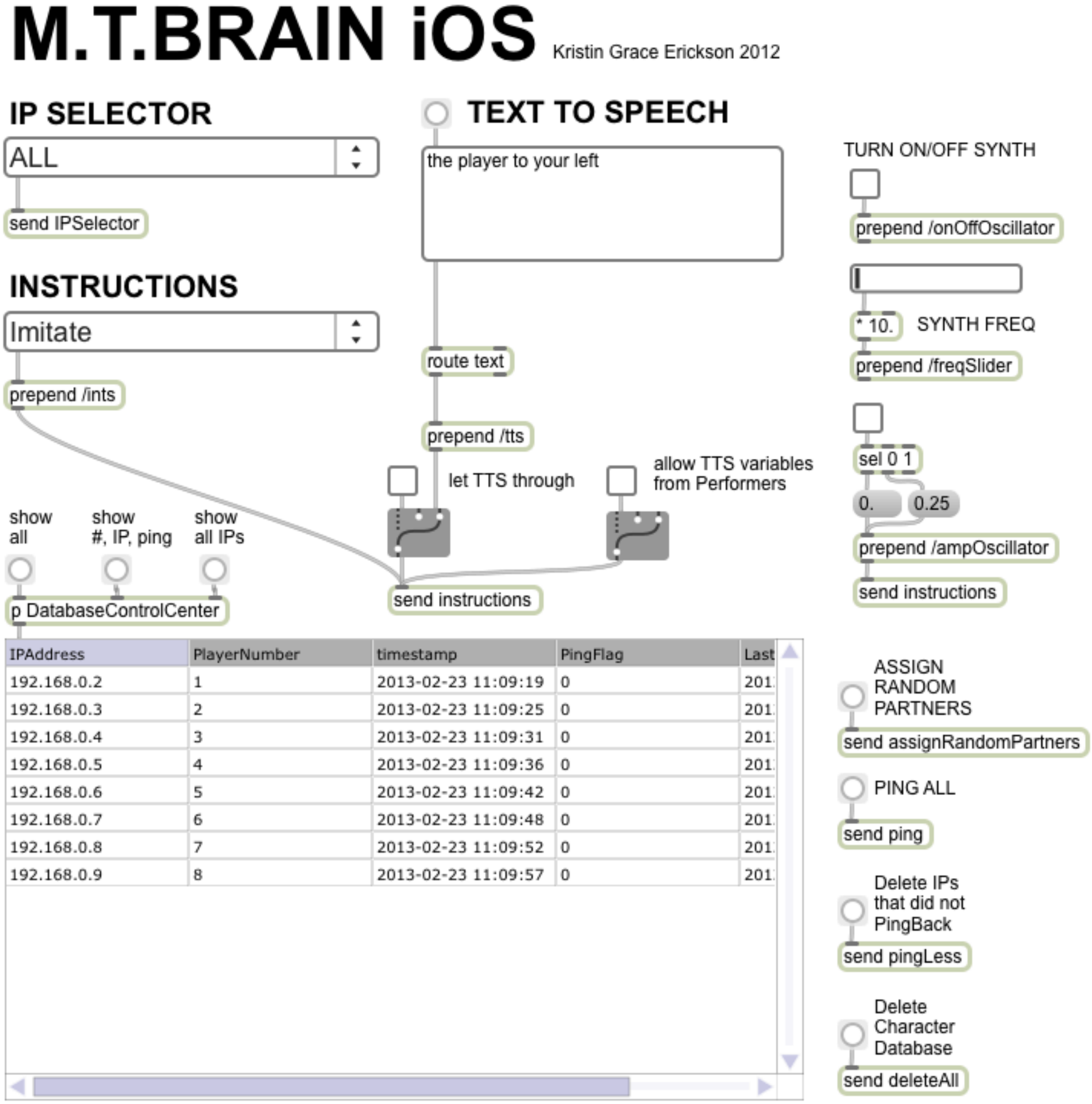
For theatrical performance, the dialogue and stage directions of a play, prewritten or prompter-improvised, are individually routed to each character. One of the audio channels is used to cue a lighting engineer while two other audio channels are routed to an amplification system for stereo sound effects and incidental music to be heard, timed with theatrical action.

Despite an abundance of potential applications spanning disciplines, the long cables connecting the sound card to the performers' headphones are a serious physical obstacle. As performers move through the space, they constantly wrangle their cables to minimize tripping hazards. The cables tangle with every movement into more complicated knots. M.T.Brain's overall aesthetic emerged as an web of wires ensnarling performers wearing large numbers.

M.T.Brain iOS

The M.T.Brain iOS app replaced cumbersome cables with wireless communication. The Max/MSP patch routes OSC (Open Sound Control) messages through a local wireless network to trigger text-to-speech audio and prerecorded audio files stored in the iOS application, as shown in Image 5.

Image 5. M.T.Brain iOS Max/MSP Interface



IP SELECTOR: ALL

INSTRUCTIONS: Imitate

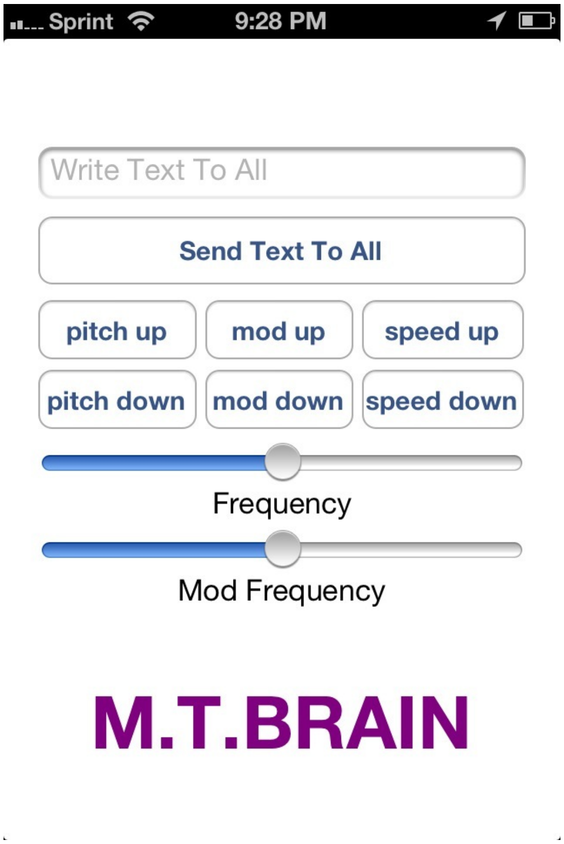
TEXT TO SPEECH: the player to your left

IPAddress	PlayerNumber	timestamp	PingFlag	Last
192.168.0.2	1	2013-02-23 11:09:19	0	201
192.168.0.3	2	2013-02-23 11:09:25	0	201
192.168.0.4	3	2013-02-23 11:09:31	0	201
192.168.0.5	4	2013-02-23 11:09:36	0	201
192.168.0.6	5	2013-02-23 11:09:42	0	201
192.168.0.7	6	2013-02-23 11:09:48	0	201
192.168.0.8	7	2013-02-23 11:09:52	0	201
192.168.0.9	8	2013-02-23 11:09:57	0	201

The iOS app includes an AM synthesizer to support instructions requiring pitch information. Audio is heard through the device's built-in speakers or headphone output.

Although wirelessly distributing OSC data through a network introduced inconsistent latency limiting the rhythmic accuracy available in the hard-wired M.T.Brain, the iOS app expanded existing M.T.Brain functionality with a user interface allowing each performer to contribute instructions during a performance, as shown in Image 6. The updated Max/MSP brain can run as a background OSC server while M.T.Brain performers prompt each other.

Image 6. M.T.Brain iOS App Interface



M.T.Brain iOS performances explore new modes of collaborative organization brought to light by performers participating in the prompting process. Decentralizing the role of the prompter replaced the aesthetic of wrangling cables with the equally pervasive aesthetic of staring at glowing phones. Nonetheless, I wrote two pieces using the M.T.Brain iOS app for audience members to perform.

In the first piece *Icebreaker,* the Max/MSP patch randomly assigns each phone's OSC output to another phone's OSC input by associating the local IP addresses. Each performer controls the frequency and modulation rate of an AM synthesizer on a randomly assigned phone. The participants walk around listening for the phone they control by sending contours recognizable over the hum of multiple modulated sine tones. When most of the participants have identified the phones they control, the Max/MSP patch is manually triggered to reassign new partners and begin again — telematic do-si-do.

The second piece *Garbledygook* utilizes the send-to-all text-to-speech function written into the iOS app. Standing on the edges of the room, each participant types a message, one at a time around the room. The text-to-speech artificial voice interpreting the message is heard on the built-in speakers of all of the phones with varying amounts of latency. After sending a message, each player takes one step towards the center of the room.

In multiple performances of *Garbledygook,* similar patterns of performance emerged. Performers type real language at first, inevitably discovering that random patterns of letters and differing amounts of letter repetitions yields unexpected audio results from the iPhone text-to-speech utility. When participants are surprised by new sounds, the messages that follow emulate and extend the discovered technique. The communication becomes more garbled and alien with every step towards the center of the room. By the end, the participants are huddled together, typing as fast as they can, giggling while their phones semi-simultaneously spit out a strange and new collective language.

Apple’s licensing and distribution restrictions complicate the development and testing of applications intended for creative telematic collaboration. To bypass the limitations of developing for proprietary hardware, M.T.Brain was rewritten in JavaScript as an open-source browser-based application called Telebrain.

TELEBRAIN

Telebrain is an internet performatization platform for creating and distributing real-time performance instructions to multiple performers using wireless web-enabled mobile devices and computers. (http://telebrain.org)

Telebrain implements the performer programming language PerPL. Dependent on Telebrain functionality, PerPL is programmed using multi-media Content Objects, performer Role Objects, performance Venue Objects, Collections of stored associations, Multi-Role and Fractional instruction Assignments, Interface Objects, OSC Objects, timing Algorithms and instruction distribution Algorithms. The collective development of PerPL is facilitated online through Telebrain, where programmers store, organize, share and implement PerPL elements and instructions.

A complete description of Telebrain functionality is found in Appendix I. The priorities of Telebrain design are as follows:

- Ensure cross-platform and cross-browser compatibility
- Maintain a robust and extensible real-time performance infrastructure
- Create a platform for the collective development of PerPL
- Make PerPL instructions available to all users
- Protect PerPL elements and instructions from alteration or deletion
- Keep Telebrain design open to unforeseen applications and uses

PerPL instructions are distributed inside of instantiated Performance Venues that function like multi-media chatrooms. Instantiated Performance Venues enable complex patterns of PerPL instructions, represented as multi-media Content, to be distributed to designated performers in the Venue. PerPL is interpreted when performers receive and perform the instructions. Educating performers about PerPL syntax, semantics, grammar and data is the process of building a PerPL interpreter. Interpreter design evolves through iterative implementations of shared PerPL instructions.

Telebrain Functionality

Telebrain Content Objects include audio files, text-to-speech audio, images, and visually formatted texts. Telebrain programmers link to online media, upload media from local devices, or manually enter texts. Content Objects saved to Telebrain are organized into longer instructions using Collections, Associations, Assignments and Algorithms.

Content Objects are used to represent programming language elements such as variables, objects, operators, expressions, functions, assignments, conditionals, statements, and scope. Content Objects can also function as data, signifying preexistent meaning(s) of sounds, images, and words. An audio recording of a cat's meow can represent a variable or operator in PerPL, or the same audio file can be used as data representing a cat, a meow or a frequency contour.

Telebrain Collections organize associations and constraints. Folders group unordered content of any type. Audio-Image Pairs link Audio Objects with Image Objects simplifying distributions to multiple receivers with varying receive settings, or simultaneously distributing Audio and Image Objects to individual receivers. Audio Sentences are sequences of Audio Objects concatenated and saved as new audio files. The start and end times of the Audio Objects in an Audio Sentence are used to reorder Audio Sentences after being served to client devices. Image Phrases are sequences of Image Objects, incremented manually or by timing Algorithms during a Performance.

Role Objects determine the functionality of a performer's mobile device limiting the types of information performers can send and receive. Interface Objects are buttons, menus, and inputs associated with Objects, Collections, and Algorithms for customizing the layout and functionality of a performer's user interface.

Assignments simultaneously distribute unique instructions to multiple performers, an implementation of MIMD processing. Multi-Role Assignments associate PerPL instructions with specified Roles. Fractional Assignments associate PerPL instructions with fractional subgroups of performers.

Telebrain's default timing delivers Content as soon as possible, but the time required for Content to arrive on wireless clients cannot be accurately predicted. For messages to arrive on multiple wireless devices at the same time, a short delay is added to all instructions accommodating for differing speeds of delivery. Content received on client devices is held until a particular tick of the internal clock. Since the client clocks can be synchronized to Telebrain's server clock, previously served instructions can be executed simultaneously after compensating for network delays.

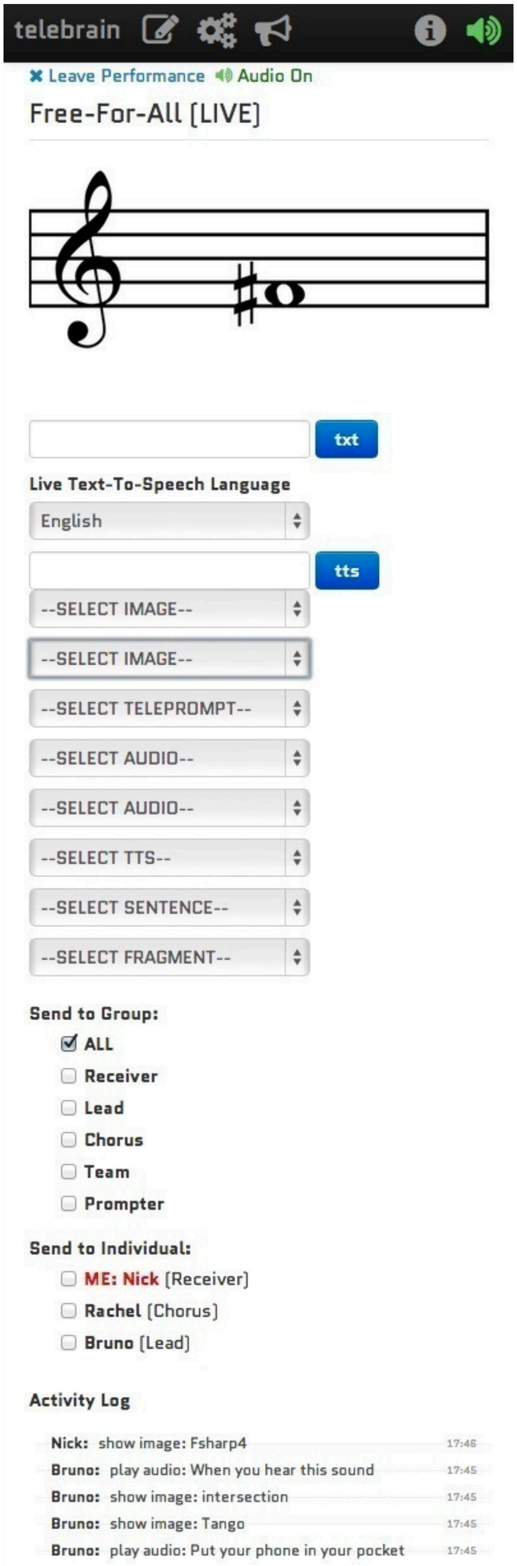
Telebrain Algorithms associate Timers, Metronomes, and programming logic with Content, Collections, Roles, and Assignments. The Timers and Metronomes control when images are displayed and audio is played. Algorithms allow OSC addresses to be associated with Telebrain functionality. External software can contribute to Telebrain Performances and external devices can receive OSC messages in response to Telebrain Performance activity.

Performing on Telebrain requires Roles and a Venue to be defined in advance. The first performer instantiates and names a Performance Venue, selects a Role, enters a nickname, and begins the Performance. Subsequent performers join the instantiated Performance Venue to participate in the Performance. Depending on Role definitions and associated Interface Objects, a Performance interface is rendered featuring lists of Content, Collections, Associations, and Algorithms available for execution. In the Performer List, checkboxes appear next to the names of performers and Roles. Instructions are routed according to Performer List designations. The Activity Log lists a record of timestamped performance instructions distributed during a Performance.

Performers in an instantiated Performance Venue can be in the same physical space as other performers or connected remotely through the Internet. Performances in multiple physical locations can occur within the same instantiated Performance Venue. Performers can leave and return to instantiated Performance Venues, but when the last performer leaves, the instantiated Performance Venue is destroyed.

As shown in Image 7, a screenshot of an active performer interface indicates the Performance is called *Free-For-All*, the performer's nickname is *Nick,* and Nick is performing a *Receiver* Role. Nick has sent the *F#4* Image Object to ALL performers including himself. A performer named *Bruno* is playing a *Lead* Role and Bruno previously sent Audio and Image Content to Nick.

Image 7. Telebrain Performance Interface Example



Free-For-All [LIVE]

Live Text-To-Speech Language

English

Send to Group:

- ☑ **ALL**
- ☐ **Receiver**
- ☐ **Lead**
- ☐ **Chorus**
- ☐ **Team**
- ☐ **Prompter**

Send to Individual:

- ☐ **ME: Nick** [Receiver]
- ☐ **Rachel** [Chorus]
- ☐ **Bruno** [Lead]

Activity Log

Nick:	show image: Fsharp4	17:46
Bruno:	play audio: When you hear this sound	17:45
Bruno:	show image: intersection	17:45
Bruno:	show image: Tango	17:45
Bruno:	play audio: Put your phone in your pocket	17:45

In addition to device memory, Telebrain performers function as memory. Like the Memory Cues in John Zorn's *Cobra,* Telebrain instructions can assign the current state of a performance to a Content Object, recalled by triggering the Content Object. Performer memory utilizes human memory capabilities and can be extended using external memory, such as writing information down on a piece of paper. Performers carrying a calculator or abacus can facilitate performatizations requiring accurately computed results from individual performer interactions.

PerPL (Performer Programming Language)

PerPL and Telebrain are designed to maximize configurability — operative as both specialized tools for specific applications and pliant tools for open-ended applications. Homologous with the Integrated Development Environments (IDEs) and language-aware text editors commonly used in computer programming, Telebrain governs the ways instructions are input, organized and delivered to PerPL interpreters. For each performance, protocols of PerPL syntax and semantics are agreed upon by PerPL programmers and Telebrain performers. PerPL's freely assignable syntax and semantics are informally stipulated through heuristic design, formed through Telebrain's infrastructure and performer capacity.

Explaining the execution of PerPL instructions to performers is the process of building multiple parallel PerPL interpreters. Training routines programmed in PerPL elucidate intended performance behavior and establish the meaning of shorthand instructions. Training can happen during warm-up routines prior to performing or training portions can be scattered throughout a Performance, introducing new protocols or shifting current rules mid-performance.

In Telebrain's Performance architecture, programmers can be performers, and vice versa.

Performances including instructions written by multiple programmers may introduce conflicting rules of syntax. When instructions are sent by the performers, for the performers, each training routine and its related instructions must indicate an association.

PerPL programmers must reinvent programming conventions to optimize human interpretation according to implementation. Techniques for indicating syntax, semantics, parsing, and scheduling require explicit definition and explanation for every Performance. Hidden, low-level programming conventions must be brought to the surface to enable high-level instructions to be performed.

In computer programming, quotation marks are often used to delimit strings. In PerPL, when devising string-delimiting techniques, the technique definition and an explanation of its function must be interpreted by the performer prior to its first use. In the following example, opening and closing quotations marks are assigned to Audio Objects. The actual quotation marks used in the example indicate phrases spoken by a text-to-speech voice.

```
//Quotation Mark PerPL Pseudocode

QuotationTraining = “When you hear this sound” + <<BEEP>> +
                    “vocally imitate what you hear next” +
                    “When you hear this sound” + <<BLEEP-BLOOP>> +
                    “stop imitating”;

QuotationPractice = “Walk to the middle of the room” +
                    <<BEEP>> + “Jump up and down” + <<BLEEP-BLOOP>> +
                    “Look left” +
                    <<BEEP>> + “Now I will do it” + <<BLEEP-BLOOP>> +
                    “Jump up and down” +
                    <<BEEP>> +
                      <<(Sung)I am jumping up and down>> + “I will stop” +
                    <<BLEEP-BLOOP>> +
                    “Stop jumping” + “Look right” + “Walk out the door”;
```

The sounds `<<BEEP>>` and `<<BLEEP-BLOOP>>` indicate if a performer jumps up and down, says the words “Jump up and down,” or jumps up and down while singing the words “I am jumping up and down.”

Performers have varying response times to different types of instructions. Some performers act as soon as they begin to understand an instruction, while other performers wait for the entire instruction before acting. By incorporating sound-triggers into instructions, performer actions can synchronize.

Two-part sound-triggers provide preparatory timing clues for imminent actions. As an audio equivalent of seeing a conductor raise her baton before a downbeat, two-part sound-triggers contain audible *preparation* and *ictus* conducting gestures. To implement `Two-part Sound-Trigger Training`, the programmer uploads a two-part sound `<<shaa-CHUNK>>` into an Audio Object and then uploads the second half of the two-part sound `<<CHUNK>>` into another Audio Object. `<<CHUNK>>` is only used during the training routine to illustrate the sound of the *ictus,* the downbeat of the two-part sound-trigger.

```
//Two-Part Sound-Trigger Training

SoundTriggerTraining = “When you hear this sound” + <<shaa-CHUNK>> +
                       “Clap your hands at the exact moment you hear the” +
                       <<CHUNK>> + “portion of the sound”;

//Two-Part Sound-Trigger Practice

      <<shaa-CHUNK>> [pause: 1000ms]
      <<shaa-CHUNK>> [pause: 8000ms]
      <<shaa-CHUNK>> [pause: 600ms]
      <<shaa-CHUNK>> [pause: 400ms]
      <<shaa-CHUNK>> [pause: 200ms]
      <<shaa-CHUNK>> [pause: 100ms]
      <<shaa-CHUNK>> [pause: 50ms]
      <<shaa-CHUNK>>
      <<shaa-CHUNK>>
      <<shaa-CHUNK>>
```

PerPL Audio Objects can function as continuous data controllers. As specifiable characteristics of a sound change through time, the contours can be correlated to the parameters of a performance. To illustrate, seven tap dancers receiving audio from Telebrain are instructed to change the speed of their tapping according to the fluctuating frequency of a sine tone instruction. In the following PerPL pseudocode, `<<(sine)__/~~~\_/\/~~~~~~~\____/\/\/\/\/~~~~~~~~~>>` is used to represent a sine tone with a fluctuating frequency contour. The underscores, slashes, tildes and backslashes represent low, rising, high and falling frequency, respectively.

```
//Frequency-to-Tap-Dance Training

FreqTapTraining = “When you hear a sine tone” + <<(sine)>> +
                  “listen to its frequency” +
                  “When the frequency is high, tap fast” +
                  “When the frequency is low, tap slow” +
                  “As the frequency fluctuates between high and low
                  adjust your dancing speed according to the changes” +
                  “When there is silence, freeze”;

//Frequency-to-Tap-Dance Practice

<<(sine)______~~~~~~~~~___/\___~~~~~~>> [pause: 500ms]
<<(sine)___\_/~~~\/~___\_/\_/~~~\____________>> [pause: 1000ms]
<<(sine)____/~~~~~>> [pause: 500ms]
<<(sine)____/~~~~~____/~~~~~>> [pause: 2000ms]
<<(sine)__/~~~\_/\/~~~~~~~\____/\/\/\/\/\\\\////~~~~~~~~~>>
```

Audio Objects and Image Objects are characterized by inherent differences when communicating PerPL instructions. Audio Objects are finite, time-dependent representations requiring subsequent Audio Objects to be concatenated or scheduled. Image Objects are instantaneous, infinitely static representations requiring subsequent Image Objects to be manually triggered or scheduled. Interrupted Audio Objects risk obscuration when instruction information is incomplete. Image Objects always render complete information and risk obscuration when the human processing the image is interrupted.

Audio instruction timing depends on the length of the audio instruction relative to how the audio communicates meaning: words, sounds, contour, transients, associations, music or patterns. Audio instruction length and efficiency must perpetually balance with the time required to perform the instruction. Rapid performance changes require succinct audio instructions to indicate the changes as fast as they are performed. Triggering new audio instructions before performers have completed previous instructions shifts the balance in the other direction. Harnessing tighter isomorphic relationships between the process-describing characteristics of audio instructions and the processual qualities driving performatizations will enhance timing stability.

Image Object timing is determined by human interpretation capacity relative to the image's function in PerPL. An Image Object of music notation may require each note to be slowly performed. The same image of music notation can come and go in a flash provoking a fleeting notion of music before boots, arrows, and popsicles are seen. An Image Object of only red may cue a sudden gesture before a green image is displayed cueing a different gesture. The same red image can function as an indefinite temporal marker for performing a deep introspective consideration of the color red.

Performatized Branching

The appropriate strategy for programming a conditional statement in PerPL relies on the nature of its impelementation. Imagine 50 performers either sitting or standing. All of the performers receive the same audio instructions from Telebrain, an implementation SIMD processing. The PerPL programmer wishes to send the following conditional statement:

```
if (standing) sit;
else stand;
```

The simplest solution is to save the above pseudocode as a Text-To-Speech Audio Object named `Stand/Sit-Switch`. Since the pseudocode is understandable in everyday language, the instruction needs no translation. When `Stand/Sit-Switch` is distributed during a performance, all performers hear the same spoken text instructing the standing performers to sit and the sitting performers to stand.

Sending `Stand/Sit-Switch` several times in a row toggles the performers between states of sitting and standing, regardless of their initial state. If a PerPL programmer plans to use an instruction repeatedly during a performance, the programmer can assign the instruction to a sound, as follows:

```
<<BLEEEEP>> =  if (standing) sit;
               else stand;
```

The programmer creates the Uploaded Audio Object `Bleeeep` by uploading the sound `<<BLEEEEP>>` to Telebrain. The text, “When you hear this sound,” is saved as a Text-To-Speech Audio Object named `When`. To build an assignment training instruction, the programmer concatenates several Audio Objects into an Audio Sentence named `BleeeepTraining.`

```
BleeeepTraining  =  When + Bleeeep + Sit/Stand-Switch;
```

Prior to using `Stand/Sit-Switch` in a Performance, the programmer sends the `BleeeepTraining` instruction. Later, when the programmer wishes a `Stand/Sit-Switch` to be performed, only `Bleeeep` is sent. Assigning functions to sounds contributes to timing precision by clarifying when instructions are performed and allowing subsequent instructions to follow in quicker succession.

If “sit” and “stand” are used in conjunction with “spin once,” “spin twice,” “squat while raising your right arm” and “touch your toes,” then an alternate programming strategy is preferred. With training, performers can learn to associate the six actions with six sounds. According to research in sonification and auralization, associating data, or an action, with the semiotic meaning of a sound contributes to learning and retention.[8]

To implement the new strategy, the PerPL programmer uploads six sounds into Audio Objects and saves descriptions of the six actions into Text-To-Speech Audio Objects. The pseudocode representing the uploaded sounds are delimited by double angle brackets. The texts within the double angle brackets are onomatopoeic representations of how the Audio Objects sound, illustrating semiotic associations with the actions they represent.

```
//6-Actions Training

6-ActionsTraining = “When you hear each sound” +
                    “perform the action that follows” +
                    <<SHLOOEEP>> + “stand” +
                    <<PLEEOOSH>> + “sit” +
                    <<SHWISH>> + “spin once” +
                    <<SHWISHWISH>> + “spin twice” +
                    <<BRAAMFDING>> + “squat while raising your right arm” +
                    <<WEWAWOWU>> + “touch your toes”;

//6-Actions Practice

“While each sound plays, perform its action”  [pause: 3000ms]
<<SHLOOEEP>> [pause: 1000ms]
<<PLEEOOSH>> [pause: 1000ms]
<<SHWISH>> [pause: 1000ms]
<<SHWISHWISH>> [pause: 1000ms]
<<BRAAMFDING>> [pause: 600ms]
<<WEWAWOWU>> [pause: 600ms]
<<BRAAMFDING>> [pause: 600ms]
<<SHWISHWISH>> [pause: 3000ms]
<<PLEEOOSH>> [pause: 2000ms]
<<SHLOOEEP>> [pause: 5000ms]
<<SHWISH>> [pause: 500ms]
<<WEWAWOWU>>
```

To integrate `6-Actions` with `Stand/Sit-Switch`, additional sounds are associated with `if` and `else`. The programmer assigns `if` to a 1000Hz sine tone. To implement an `if` statement, the sound of an action is understood as a comparison because it is overlaid with a

1000Hz sine tone — the sound of the action under evaluation and the 1000Hz sine tone are heard at the same time. To implement `Stand/Sit-Switch`, `<<SHLOOEEPP>>` will represent both the state of *standing* and the instruction to *stand*. To test *if* a performer is *standing,* the sound `<<SHLOOEEPP>>` is played simultaneously with a 1000Hz sine tone. The next sound played represents the action to perform in the first branch of the conditional statement.

To implement the `else` branch of the conditional statement, `else` is assigned to the sound `<<CLUNK>>`. Since `else` does not require a comparison, the sound is played unlayered. The sound that follows `<<CLUNK>>` represents the action to perform in the second branch of the conditional statement.

PerPL programmers can layer audio by saving Audio Objects into an Audio Layer Object. In the example below, the pseudocode `LAYER [ AudioObject1, AudioObject2 ]` is used to represent an Audio Layer Object.

```
//If-Else Training

If-ElseTraining = “if you hear the sound for an action” +
                  “overlaid with this sound” + <<1000HZ SINE TONE>> +
                  “ask yourself if you are performing the action” +
                  “if you are” +
                  “perform the action for the next sound you hear” +
                  “if you are not” +
                  “do not perform the next sound you hear” +
                  “when you hear the sound” + <<CLUNK>> +
                  “if you did not perform the previous sound” +
                  “perform the action for the next sound you hear”;

//If-Else-Practice

LAYER [ <<1000HZ SINE TONE>>, <<SHLOOEEP>> ]
<<PLEEOOSH>>
<<CLUNK>>
<<SHLOOEEP>>

LAYER [ <<1000HZ SINE TONE>>, <<PLEEOOSH>> ]
<<SHLOOEEP>>
<<CLUNK>>
<<PLEEOOSH>>
```

```
LAYER [ <<1000HZ SINE TONE>>, <<SHLOOEEP>> ]
<<SHWISH>>
<<CLUNK>>
<<BRAAMFDING>>

LAYER [ <<1000HZ SINE TONE>>, <<BRAAMFDING>> ]  // if(a){b} if(c){a} else{d}
<<WEWAWOWU>>
LAYER [ <<1000HZ SINE TONE>>, <<SHWISH>> ]
<<BRAAMFDING>>
<<CLUNK>>
<<SHWISHWISH>>

LAYER [ <<1000HZ SINE TONE>>, <<BRAAMFDING>> ]
<<SHLOOEEP>>
<<CLUNK>>
<<PLEEOOSH>>
```

Limiting performatizations to conditionals and actions can generate complex results. `If-Else-Practice` illustrates the use of PerPL instructions made entirely from non-speech sounds. As PerPL instructions become more complex, PerPL intepreters and their training routines develop as well.

At which point will PerPL audio instructions be repurposed as music?

THE FUTURE

The future of Telebrain is divided into immediate goals of increased functionality and distant glimpses of what could become.

Immediate Future

The immediate goals of Telebrain extend current functionality, stimulate PerPL development, and incorporate potent research from other fields. Future Telebrain developments are listed as follows:

- **Add sound synthesis -** A browser-based sound synthesis engine reduces the latency of serving audio files in real-time. Greater varieties of modifiable sound will augment meaning transmission potential. Currently on hold until robust, cross-platform libraries are available.
- **Implement spearcons**[9] - Spearcons are time-compressed words or phrases that are played too fast to be recognized. Developed for auralization, spearcons outperform regular speech in experimental data. Since they are lexical, they exploit the language processing centers of the brain. Currently on hold until a free, variable-speed, cross-platform text-to-speech solution is available. The current text-to-speech implementation resolves compatibility issues that arise with the variable-speed alternatives.
- **Utilize built-in client device features** - Most computers and mobile devices include GPS, accelerometers, built-in microphones, built-in cameras, text-to-speech utilities and the ability to vibrate. Future implementation depends on cross-platform browser access to these features.
- **Improve usability and explainability** - To maximize potential uses and users, emphasize user-centered development by making clear and simple interfaces for complex processes. Advocate Telebrain’s use across disciplines.
- **Simulate evolutionary dynamics in PerPL's development architecture** - Rebuild the the PerPL instruction programming architecture to capitalize on existing evolutionary dynamics of human social systems. Investigate evolutionary algorithms appropriate for network-based collective language development and model the processes into Telebrain’s PerPL development architecture.

- **Automate PerPL instruction generation** - Implement user-definable Telebrain templates for automatically generating generic performatization schemes. Performance architectures defined by network topologies of programmer/performer relationships can be preset. Using commonly implemented PerPL syntax and semantic rules to automatically generate PerPL instructions will simplify the programming process — limiting configurability when desired. Templates invite new PerPL programmers to experience Telebrain with less investment. Advanced PerPL programmers can develop templates for reusable performance architectures and PerPL rules.
- **Develop interfaces for isomorphic transference** - Implement sonification and auralization techniques for generating PerPL instructions. Develop interfaces for easy translations between datasets and algorithms into PerPL instructions.
- **Create a text-based version of PerPL** - Design a Telebrain text editor for a specified, extensible text-based PerPL. Text-based PerPL statements and codeblocks will be automatically translated into multi-media PerPL instructions. Incorporate schemes for converting text-based PerPL into the grammar of spoken language, and vice-versa.
- **Integrate models of human cognition** - Model the neurobiological timing thresholds of the human listening process into parameters of PerPL instructions. Focusing, aiming, or distributing heirarchies of algorithmic processes into specified temporal ranges of aural perception will enhance the interpretability of audio instructions.
- **Incorporate principles of ABL (A Behavior Language)** - ABL is an artificial behavioral programming language used to control digital autonomous agents in virtual interactive environments.[10] Making artificial interactions seem real, behavioral programming

languages are primarily used in virtual environments and video games. Applying principles of ABL to real-time, real-space, real-human performances may reveal new techniques for organizing performance instructions. ABL supports joint action by systematically coordinating the behavior of multiple performers.[11] Future actions of performers depend on the results of current performer actions. The following is an example of ABL code implemented when there is a knock at the door:

```
sequential behavior AnswerTheDoor() {
    WME w;

    with success_test { w = (KnockWME) } wait; act sigh();
    subgoal OpenDoor();
    subgoal GreetGuest();
    mental_act { deleteWME(w); }
}

sequential behavior OpenDoor() {
    precondition { (KnockWME doorID :: door)
          (PosWME spriteID == door pos :: doorPos)
          (PosWME spriteID == me pos :: myPos)
          (Util.computeDistance(doorPos, myPos) > 100)
    }
    specificity 2;
    // Too far to walk, yell for knocker to come in subgoal
    YellAndWaitForGuestToEnter(doorID);
}

sequential behavior OpenDoor() {
    precondition { (KnockWME doorID :: door) }
    specificity 1;
    // Default behavior - walk to door and open
}[12]
```

Distant Future

Continuous integration of heterogeneous configurations fuels the locomotion towards the boundary of imaginations. Communication is the medium — where signals signal signals. Communication as computation, literally conceived as metaphor, is elaborated into the future using vibrational computation.

Historically, proof of artificial intelligence is thought to manifest in conversations between human and machine.[13] When the conversation is unsatisfactory, half of the conversation is physically reconfigured to dictate a different conversation.

Human symbolic language co-evolved with paralinguistic prosodic capabilities. Equivalent paralinguistic functions are rarely found in artificial symbolic languages such as mathematics and computer programming languages. In contrast, the artificial symbolic languages used to notate music represent parameters of sound that correlate with paralinguistic characteristics. Musical expression, often changes in volume, rhythm and pitch, can provoke internal experiences considered difficult to describe with words.

The space between language and music is an area of underutilized capacity where current modes of communication can expand. Due to the tight evolutionarily entanglement between our hyper-sensitive oral-aural capacity, cognitive infrastructure and language facility, sound is a logical nursery for growing fresh branches of communication. Since sound is carried by waves, a future computational paradigm is imagined as intersecting vibrations.

Vibrational computation occurs when waves co-compute midstream. The intersection of computing waves outputs interference pattern results. A multiplexed wave embodying program, data, interpreter, and self-describing co-signal-processing ability intersects with a similarly data-enriched wave. Always containing incomplete information, each signal relies on interaction to fill in the gaps. As the waves intersect, the combined instructions interlock. Through superimposed micro-interference, canceling and reinforcing as they perpetually go, the multiplexed waves co-interpret each other.

The evolution of symbolic language distanced humans from their subjective experiences. Once experiences were untethered from experiencer, they could be interpreted, shared and reconfigured — recursively filtered through generations of brains.

Vibrational computation places interpretation in the space between communicators, creating actual interpretive distance. The external interpreter is conjured by the act of communication — as if each signal contained the genetic information for creating an instantaneous brain at every shared intersection. Vibrational computation resonates through self-describing wave-driven zippers interlocking patterns like gamelans through space.

CONCLUSION

Telebrain is a platform for entangling systems of meaning, a place where paradoxes self-resolve through the evolutionary potential introduced by incorporating human interaction into the computational process.

Computers evolved due to generations of humans externalizing their internal cognitive structures.[14] Holding up a labyrinthian mirror, Telebrain externalizes the computer's internal cognitive structures into novel paradigms of human performance. Communication is the common thread, where incongruous systems of logic, language and art commingle, configure, and transform.

Appendix I. Telebrain Functionality (http://telebrain.org)

CONTENT OBJECTS

Audio Objects

Web-Based Audio: Web-based audio allows online audio files to be available during a performance. Copyrighted material is not allowed. When saving Web-Based Audio Telebrain makes a local MP3 copy of the audio on the Telebrain sever. Currently, only audio file types supported by the user's browser are supported.

Uploaded Audio: Audio uploads copy audio content from a client device to the server. Only MP3 audio files are supported at this time.

Text-To-Speech Audio**:** Text-To-Speech audio can be saved in advance or generated in real-time during a performance. To make Text-To-Speech audio in advance, choose a language and then save up to 100 characters of text. When the save button is pressed, Telebrain saves an mp3 of the Text-To-Speech audio to the server. Text-To-Speech audio can be accessed during a performance or concatenated with other audio using the Audio Sentence functionality.

Audio Collections**:** Since Audio Sentences and Audio Layers generate a new audio files when saved, these Collections function as Audio Objects. Audio Sentence and Audio Layers can be incorporated into new Collections, however altering the original Audio Objects after they are used in Audio Sentences or Audio Layers does not ensure that preexisting Collections will be updated. Generally, new audio files are generated at the time the Collections are saved to Telebrain - re-saving the referencing Collections updates the audio files to the current version. This may be automated in future versions of Telebrain.

Image Objects

Web-Based Images: Web-based images allow graphic internet content to be made visible during a performance. To save a web-based image, find a valid URL linking directly to online image content. A valid image URL with end with '.jpg' or '.png', and should not be a link to an html page containing the image. The easiest way to obtain a valid image URL is to right-click on the desired image in order to open the image in a new tab or window

Uploaded Images: Image Uploads allow content uploaded from a computer or mobile device to be available during a performance. The upload functionality is currently suspended. In the meantime, email the image to Telebrain and the image will be uploaded manually.

Teleprompts: (Text-To-Image) Teleprompts graphically display text during a performance. Parameters such as font, size, text color, and background color can be assigned to each Teleprompt Object.

COLLECTION OBJECTS

Folder Objects: Folders are unordered collections of Telebrain Content and Programs. Folders are an organizational tool and allow associated data to be assigned to particular Roles during a performance. Folders can be used to filter content used in a particular performance and can also be used in Timed Organization Algorithms.

Audio-Image Pairs: Audio-Image Pairs allow Image Objects and Audio Objects to be distributed simultaneously to a Role during a performance. Audio-Image Pairs function similarly to other Telebrain Content and automatically adjust to Roles with limited Audio / Image receiving functionality.

Audio Sentence Objects: Audio Sentences are ordered collections of Audio Objects that are concatenated into a single audio file. Audio Sentence create a new concatenated audio file in order to reducing the timing inconsistencies of delivering multiple smaller audio files individually. The start times of individual audio files in a generated Audio Sentence can be used to quickly change the order the Audio Sentence is played after the audio is delivered to the performers. When a new Audio Sentence is saved, the Collection can be used as an Audio Object.

Audio Layer Objects: Audio Layers allow Audio Objects to be played simultaneously during a performance. Each Audio Object layer requires an assigned start-time and relative volume level. The Audio Objects in an Audio Layer are mixed and saved to a new audio file. When new Audio Layers are saved, the Collection can be used as an Audio Object.

Image Phrase Objects: Image Phrases are ordered Collections of Image Objects available for use during a performance. Image Phrases can be stepped through manually during a performance or used in Organization Algorithms.

PROGRAMS

Program Setup

Role Objects: Roles organize possible performance functions allowing each performer or performer group to have their send/receive capabilities and interface layout assigned in advance. Multiple performers can play the same role in a performance. The maximum number of performers per Role can be limited in the Venue model. Checkboxes turn the following visibility and functionality on or off for each Role: Telebrain Menu, Performance Title, Send Text, Send Text-To-Speech (live), Send Image, Send Audio, Send Association, Send Fraction, Send OSC, Receive Text, Receive Text-To-Speech (live), Receive Image, Receive Audio, Receive Interface, Receive OSC, Role List, Performer List, Performer Activity Log, Global Activity Log, Change Role, Change Interface, Change Functionality, Test Functionality. See Performance Functionality for more information

Venue Objects: Venues are models of performance architecture. The parameters assigned to each venue will determine how Roles interact in a performance. When performing, a new instance of the venue model is loaded allowing performers to join by selecting a Role with predetermined functionality. Venue models may need to be declared before creating Associations, Algorithms, and Content assignments depending on the interrelational requirements of a particular performance. Venue declarations can limit the number of performers allowed per Role. The type of information needed when joining a performance is determined such as requiring a nickname, local IP address, or passcode.

Interface Objects: Interfaces allow Content, Collections, Associations, and Algorithms to be assigned to user interface elements such as buttons, pull-down menus, text inputs, and display areas. Interfaces can be dynamically rendered, associated with Roles, and/or incorporated into Collections, Associations, and Algorithms.

Assignments

Multi-Role Assignments: Multi-Role Assignments allow Content to be associated with multiple Roles in advance. Multi-Role Assignments can distribute a variety of Content simultaneously to different Roles during a live performance. Multi-Role Assignments can be used in Timed Organization Algorithms in order to construct other multi-performer distribution presets. Multi-Role Assignments are dependent on a pre-existing venue because the available roles must be known in advance. Multi-Role Assignments are also dependent on the functionalities assigned to each Role, because the associated roles must be able to receive the assigned content.

Fractional Assignments: Fractional Assignments associate different Content with fractions of a group or Role (unlike Multi-Role Assignments where different content is assigned to different roles). Fractional Assignments default to having ALL performers as the group to be divided, but the assignments can also be associated to a particular Role or Roles. Fractional Assignments have two modes of dividing a performer group. Persistent Mode remembers the Fractions performers are assigned to throughout the performance. Dynamic Mode randomly divides performers into new Fractions each time the Fractional Assignment is called.

Algorithms

Timers: Timers of particular lengths of time can be saved in advanced for use during a performance. Timers are necessary for synchronizing events on multiple devices due to latency inconsistencies encountered delivering data from the server to multiple clients. Telebrain timers are self-adjusting and continually synchronize to the server clock in order to assure time-accuracy. Although immediately synchronized events cannot be guaranteed, events can be triggered simultaneously after a predetermined amount of delay.

Metronomes: Metronomes allow multiple performance events to be triggered at a regular timed interval. Multiple Metronomes can be synchronized to the server or run at varying tempi without synchronization.

OSC Objects: OSC (Open Sound Control) Objects allow incoming and outgoing OSC address parameters to be associated with Telebrain Content, Collections, and Programs. OSC Objects allow Telebrain to interface with external OSC-capable hardware and software, and control information to be routed to clients through a local network bypassing the remote Telebrain server.

Timed Organization: Timed Organization Algorithms allow Timers and Metronomes to be assigned to OSC Objects, Content, Collections, Roles, Venues, Interfaces, and Associations. This is where timing functions can be assigned to algorithmically organize all other Telebrain functionality.

Distribution Organization: Distribution Organization Algorithms allow OSC Objects, Content, Collections, Roles, Venues, Interfaces, and Associations to algorithmically organized without timed restriction.

PERFORMANCE

Start Performance: To start a new Performance, the first performer must instantiate a Venue model by selecting from a list of Venues and naming the Performance. Once the first performer has chosen a Role and provided a nickname and/or IP address, if required, the Performance exists and becomes available for additional performers to join. The Performance can be protected by an associated passcode allowing only performers with the correct passcode to join.

Join Performance: If a Performance exists, a "Join Performance" pull-down menu listing current Performances is rendered allowing performers to select the Performance to join. The performer selects a Role and if required, enters a unique nickname and/or their local IP address.

Performance Functionality

Telebrain Menu: Show or Hide the Telebrain Navigation menu. Hiding creates a larger display area for Image Content and Interface, but limits Telebrain navigation during a Performance.

Performance Title: Show or Hide the name of the current Performance. Hiding allows more display area for Image Content and Interface, but multi-Venue Performances may require performers to be aware of their current Performance name.

Send Text: Show or Hide text input interface for typing and sending real-time text during a Performance. Send Text functions as a chatroom text input and can be routed to some or all performers depending on routing assignments and the receiving Roles' functionality.

Send Text-To-Speech (live): Show or hide text input interface for typing and generating real-time Text-To-Speech Audio during a Performance. The Send Text-To-Speech functions as a chatroom text input, except the received text is received as Text-To-Speech Audio. The Text-To-Speech audio can be routed to some or all performers depending on routing assignments and the receiving Roles' functionality.

Send Image: Show or hide a pull-down menu listing the Image Content available to a particular Venue and/or Role. If no Image Content has been assigned, then all Telebrain Image Content is listed. When selected, the Image Content is immediately sent to some or all performers depending on routing assignments and the receiving Roles' functionality.

Send Audio: Show or hide a pull-down menu listing the Audio Content available to a particular Venue and/or Role. If no Audio Content has been assigned, then all Telebrain Audio Content is listed. When selected, the Audio Content is immediately sent to some or all performers depending on routing assignments and the receiving Roles' functionality.

Send Multi-Role: Show or hide a pull-down menu listing the Audio Content available to a particular Venue and/or Role. If no Audio Content has been assigned, then all Telebrain Audio Content is listed. When selected, the Audio Content is immediately sent to some or all performers depending on routing assignments and the receiving Roles' functionality.

Send Fraction: Show or hide a pull-down menu listing the Fractional Assignments available to a particular Venue and/or Role. If no Fractional Assignments have been associated, then all Telebrain Fractional Assignments appropriate to the Venue model will be listed. When selected, the Content associated to each Fraction is immediately sent to the performers divided into Fractions. Fractional Assignments send Audio and/or Image Content to associated performers.

Send OSC: Show or hide a pull-down menu listing the OSC Presets available to a particular Venue and/or Role. When selected, the preset OSC message is immediately sent to the local IP addresses associated with some or all performers depending on routing assignments and the receiving Roles' functionality.

Send Algorithm: Show or hide a pull-down menu listing the Algorithms available to a particular Venue and/or Role. If no Algorithms have been associated, then all Telebrain Algorithms appropriate to the Venue model will be listed. Depending on the selected Algorithm, new Interface elements may appear depending on the input requirements of the Algorithm. Control of the Algorithm will be made available to the sender.

Receive Text: Allow received text to be displayed.

Receive Text-To-Speech (live): Allow the received audio file generated from real-time Text-To-Speech text to be played on the local device.

Receive Image: Allow received Image Content to be displayed.

Receive Audio: Allow received Audio Content to be played on the local device.

Receive Interface: Allow received Interface Content to be displayed and used.

Receive OSC: Allow received OSC messages to trigger other Telebrain functionality.

Role List: Show or Hide the list of Roles associated with the Venue model of the Performance. Checkboxes are displayed next to each Role in the list to indicate the Roles to which Content will be routed. Associations, Fractions, and Algorithms override Role Routing Assignments.

Performer List: Show or Hide the nicknames of current performers and indicate their Role. Checkboxes are displayed next to each Nickname in the list to indicate the specific Performers to which Content will be routed. Associations, Fractions, and Algorithms override Performer Routing Assignments.

Performer Activity Log: Show or Hide a list of the times all Instructions Sent or Received by the Performer .

Global Activity Log: Show or Hide a list describing all Instructions Sent or Received by all Performers.

Change Role: Show or Hide a pull-down menu of Roles associated with the Venue. The performer switches to the selected Role.

Change Interface: Show or Hide a pull-down menu of Interfaces associated with the Role and/or Venue. The display shows or hides the selected Interface.

Change Functionality: Show or Hide a pull-down menu of Performance Functionalities, allowing the current Performance Functionalities to changed during a Performance.

Test Functionality: Show or Hide the necessary interface elements for Testing currently assigned Functionalities. When Testing, the performers can send themselves Content to test their receive settings or receive Content to test their send settings.Test functionalities can be controlled by a master prompter in the form of a Sound Check Algorithm or Image Check Algorithm.

Leave Performance: A link that reads “X Leave Performance” is displayed in the top-left corner of all live Performances. Clicking on this link removes the performer from the current Performance and returns to the Telebrain website.

Audio Required: If a Role indicates Audio Receive functionality is required, a link that reads “Audio Required” is displayed to the right of the “X Leave Performance” link. Clicking the “Audio Required” link turns Audio On if the audio is turned off and turns Audio Off if the audio is already on. The Speaker in the upper-right corner of the Telebrain navigation menu indicates the current Audio state as well, but may not be visible if Telebrain Menu is hidden. When Audio is switched from off to on, the “Telebrain” audio file is played.

GENERAL FUNCTIONALITY

Speaker Icon: The icon in the top-right corner of the Telebrain Navigation Menu is red when Audio is Off and green when Audio is On. When Audio is switched from Off to On, the “Telebrain” audio file is played. The Speaker Icon must be green indicating that the Audio is On for Audio Content to be played by Telebrain.

Lock/Unlock: Content, Collections, and Programs can be locked and unlocked in order to protect user-generated data. A passcode is required when locking saved data and can only be unlocked and therefore edited when the same passcode is entered. Since all data stored on Telebrain can used in any Performance, locking and unlocking data protects the information from being altered or deleted by another user.

Acknowledgments

This work was supported by the Herb Alpert School of Music at the California Institute of the Arts and benefited from the valuable input of my doctoral committee: David Rosenboom, Sara Roberts, and Robert Wannamaker.

I would like to thank Mark Trayle, Tim Perkis, Clay Chaplin, Anne LeBaron, Lauren Pratt, Andrea Young, Bill Gribble, Hugh Robert MacMillan, STEIM, WACM and the Improvised Music Theater classes at California Institute of the Arts for additional ideas and support.

References

1. Dubus, G., Bresin, R. (2011). Sonification of Physical Quantities Throughout History: A Meta-study of Previous Mapping Strategies. *The 17th International Conference on Auditory Display*, 1.
2. Frysinger, S. (2005). A Brief History of Auditory Data Representation to the 1980s. *First Symposium on Auditory Graphs*, 2.
3. Frysinger, S. (2005). A Brief History of Auditory Data Representation to the 1980s. *First Symposium on Auditory Graphs*, 1.
4. Vickers, P., Alty, J. (2006). The Well-Tempered Compiler? The Aesthetics of Program Auralization. *Aesthetic Computing,* 335-354.
5. Worrall, D. Sonification and Information: Concepts, Instruments, and Techniques. *Dissertation, University of Canberra*, 2-9.
6. Worrall, D. Sonification and Information: Concepts, Instruments, and Techniques. *Dissertation, University of Canberra,* 2-8.
7. Smith, S., Stephan, K., Parker, S. (2004). Auditory Warnings in the Military Cockpit: A Preliminary Evaluation of Potential Sound Types. *DSTO Systems Sciences Laboratory,* 1-2.
8. Scaletti, C. (1994). Sound Synthesis Algorithms for Auditory Data Representation. *Auditory display: Sonification, Audification, and Auditory Interfaces*, 224.
9. Worrall, D. Sonification and Information: Concepts, Instruments, and Techniques. *Dissertation, University of Canberra*, 2-9.
10. Mateas, M., (2002). Interactive Drama, Art, and Artificial Intelligence. *Thesis, Carnegie Mellon University,* 4.
11. Mateas, M., (2002). Interactive Drama, Art, and Artificial Intelligence. *Thesis, Carnegie Mellon University,* 74.
12. Simpkins, C. (2009). ABL Documentation. February 13, 2013, from http://www.cc.gatech.edu/~simpkins/research/afabl/ABL_Documentation.pdf.
13. Turing, A., (1936). On Computable Numbers, With an Application to the Entscheidungsproblem. *In Proc. London Math. Soc.*, **2(42)**, 230-265.
14. Bateson, G. (1972). *Steps to an Ecology of Mind* (2nd Edition). San Francisco: Chandler.

Harnessing the Intelligence of *Physarum Polycephalum* for Unconventional Computing-Aided Musical Composition

Eduardo Reck Miranda
Interdisciplinary Centre for Computer Music Research (ICCMR)
School of Humanities and Performing Arts, Plymouth University
Plymouth, United Kingdom
eduardo.miranda@plymouth.ac.uk

Abstract

This paper introduces *Die Lebensfreude*, a pioneering piece of music composed with the aid of an amoeba-like plasmodial slime mould called *Physarum polycephalum*. The composition is for an ensemble of five instruments (flute, clarinet, violin, cello and piano) and six channels of electronically synthesises sounds. The instrumental part and the synthesised sounds are musifications and sonifications, respectively, of a multi-agent based simulation of *Physarum* foraging for food. The slime mould, its simulation, and musifications and sonification methods are introduced in this paper. The rational for using *Physarum* in music is also discussed.

Keywords: Physarum polycephalum music, unconventional computing for music, sonification of slime mould, future of music technology.

1 INTRODUCTION

Physarum polycephalum, hereafter referred to as *Physarum goo*, inhabits cool, moist, shaded areas over decaying plant matter, and it eats nutrients such as oat flakes, bacteria and dead organic matter. It is a biological computing substrate, which has been enjoying much popularity within the Unconventional Computing research community for its astonishing computational properties [1]. Whilst scientists are looking into the possibility of harnessing Physarum goo's behaviour in order to build biological computers, I am interested in harnessing its behaviour to produce music.

The motivation to compose this piece emerged after a painting by German prolific artist Max Ernst, *Die Lebensfreude* (translated into English as *The Joy of Living*), from 1936, which I observed at the Scottish National Gallery of Modern Art in Edinburgh. I learned that this work is a twist on a painting by Matisse with the same name. In contrast to the joyous nature of Matisse's painting, Ernst spreads entangled leaves and tendrils across the picture and populates it with praying mantises. Lost in this sinister world is a diminutive human being alongside a crouching beast. This predatory jungle is an expression of the Ernst's outrage at the worsening political situation in Europe during the 1930s. This piece is an attempt at conveying musically the feeling that this painting elicited on me.

The composition is for an ensemble of five instruments (flute, clarinet, violin, cello and piano) and six channels of electronically synthesised sounds (Figure 1). The instrumental part and the synthesised sounds are musifications and sonifications, respectively, of a computer simulation of Physarum goo foraging for food. A visual animation of the simulation that generated the materials for the composition is displayed during the performance, but the images are twisted by the musicians as they play: the music controls software that manipulates the animations in real-time. Each instrument holds a microphone, which relays the sound to a system that controls the images.

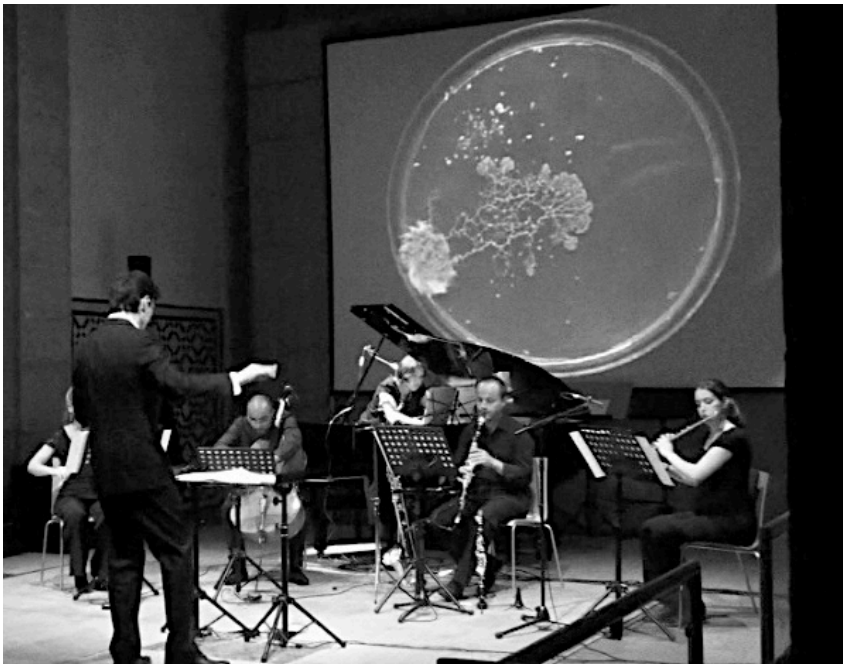


Figure 1: Performance of *Die Lebensfreude* by Sond'Ar-te Electric Ensemble, conducted by Guillaume Bourgogne, at Cascais Cultural Centre, Portugal. In addition to the musicians on stage, the piece involves six loudspeakers distributed in the concert hall (not show in this photo) to relay six channels of electronically synthesised sounds.

Die Lebensfreude has two movements: *Machina Vita* and *Machina est Finitum*. This paper concerns mostly the first movement, albeit the second movement also embodies similar concepts.

The paper starts with a discussion on the rational for using Unconventional Computing in music. Then, it introduces Physarum goo, followed by a method for sonifying its behaviour, which was used to generate the electronic sounds of the composition. Next, it presents a simulator of Physarum goo's behaviour, which forms the core of a music sequencer that is presented next. Finally, it introduces the system that was put together to compose the music, followed by an explanation of how the composition method works. The paper ends with concluding remarks on using Physarum goo to compose music and the future of Unconventional Computing in music.

2 WHY MUSIC WITH UNCONVENTIONAL COMPUTING?

Computers have been programmed to generate music as early as the beginning of the 1950's [2]. Nowadays, they are ubiquitous in many aspects of music, ranging from software for musical composition and production, to systems for distribution of music through the Internet. Therefore, it is likely that future developments in computing technology will continue to impact on the music industry. The relatively new field of Unconventional Computing [3] is no exception.

My research is looking into the development of new approaches to musical composition with computers. I am particularly interested in establishing ways in which computers can aid creativity. During the course of my research I have built a number of software systems that generate music materials, which I subsequently used in my compositions [4, 5, 6]. Generally speaking, computers have aided my creativity by generating musical materials automatically for my pieces, which I would not have produced on my own manually. These materials include riffs, sequences, rhythms, melodies, entire sections lasting for several minutes, and indeed synthesised sounds.

Technically, there are two approaches to designing computer systems to generate music, which I refer to as the *Artificial Intelligence (AI)* and the *algorithmic* approaches, respectively. The AI approach is concerned with embedding the system with musical knowledge to guide the generative process. For instance, computers have been programmed with rules of common practice for counterpoint and voicing in order to generate polyphonic music [7]. As a matter of fact, machine-learning technologies have enabled computers to learn musical rules automatically from given musical scores, which are subsequently used to generate music. The algorithmic approach is concerned with translating data generated from seemingly unmusical models onto music. Examples of this approach abound, including computers that have been programmed to generate music from chaotic functions [8], fractals [9] and cellular automata [10].

Aesthetically, the algorithmic approach tends to generate highly novel and unusual music, whereas the AI approach tends to generate imitations of certain types of music. Both approaches have their own merits and pitfalls. It is important, however, to include in this discussion how composers make use of computer-generated materials in their work. Again, I suggest two approaches here, which I refer to as the *purist* and *utilitarian* approaches, respectively.

The purist approach to computer-generated music tends to be more concerned with the correct application of the rules programmed in the system, than with the musical results per se. In this case, the output of the computer tends to be considered as the final composition. That is, the composer would not normally modify the music in this case, as this would meddle with the integrity of the model or system. At the other end of the spectrum is the utilitarian approach, adopted by those composers who consider the output from the computer as raw materials for further work. Here composers would normally tweak the results to fit their aesthetic preferences, to the extent that the system's output might not even be recognisable in the final composition. Obviously, there is a blurred line dividing these two approaches, as practices combining aspects of both are commonly found.

It is important to acknowledge these different practices in computer music in order to widen our appreciation of the aforementioned impact of computing technologies on music.

By way of related research, I cite the development of a granular synthesiser using models of reaction-diffusion chemical computing [11] and a method to sonify of the behaviour of *in vitro* neural networks [12]. As far as I am aware, my collaborators and I are pioneers in using Physarum goo to generate music.

3 PHYSARUM POLYCEPHALUM

Prototypes of novel computational devices that have been recently developed include DNA computers, reaction-diffusion chemical computers, molecular machines and bacterial computers [13]. However, these are costly to build and maintain.

Conversely, Physarum goo is a biological computing substrate, which is comparatively easier to handle and more cost-effective than those other approaches mentioned above.

A picture of Physarum goo is shown in Figure 2. The blob is a huge single cell, but unlike most cells, which have only one nucleus, this cell contains millions of nuclei. This giant cell moves like an amoeba, propagating over the surface as it ingests bacteria, leaves and rotten wood, but only pictures taken over several days can show its progress.

The main phase, and the one that holds most interest for us here, is the plasmodium phase, which forms streams of slime moulds in search of nutrients. The goo surrounds nutrients and secretes enzymes to digest it. Physarum goo will enter in a dormant stage when the environmental conditions became unsuitable. In this case it forms a protective hardened tissue referred to as sclerotium. When conditions are favourable, it reverts into plasmodium and continues its quest for nutrients. Physarum will enter in reproductive state if there are no longer nutrients in the environment. Spores are formed and released into the air to be spread by wind. Those spores might remain dormant for years, until favourable conditions make possible it to germinate and release swarm cells that fuse together to form a new plasmodium.

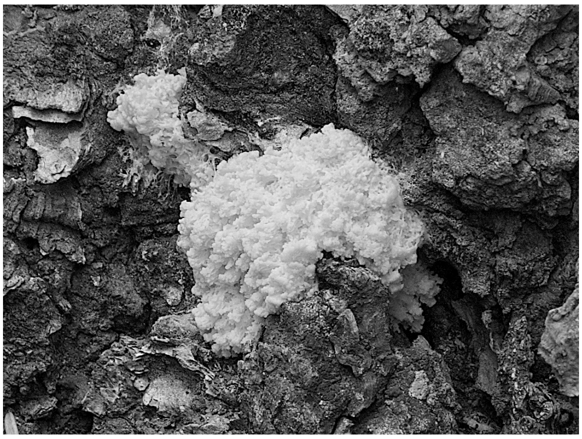


Figure 2: Physarum polycephalum (Physarum goo) is a huge single cell, but it contains millions of nuclei. (Source Wikimedia Commons, photo by J. Kirkhart.)

Physarum goo can be cultured in the laboratory by placing it on a dish with scattered sources of nutrients; e.g., oat flakes. All being well, it goes on to form a network of protoplasmic tubes connecting the nutrient sources (Figure 3). Physarum goo is an attractive candidate for research into biological unconventional computers because its behaviour is controllable: it reacts to attracting (e.g., food and humidity) and repelling (e.g., light and salt) sources. By placing repellents and attractors in the environment one can prompt the goo to behave in specific ways.

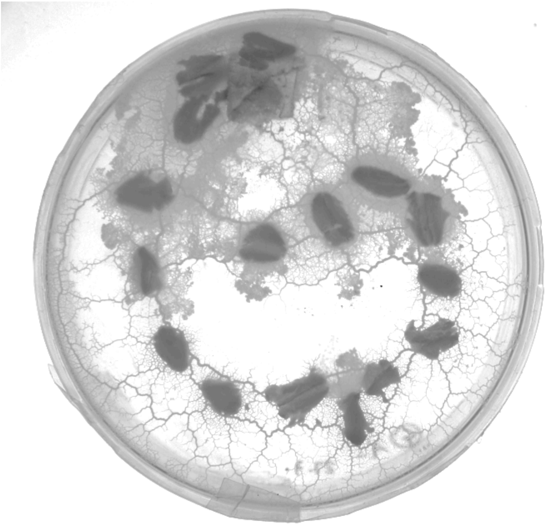


Figure 3: Physarum goo in a dish with scattered sources of nutrients. (Source: http://physarum.wordpress.com/)

It has been demonstrated on various occasions that a Physarum goo-based machine is able to perform computational tasks. There are published reports that computing devices based on Physarum goo were capable of solving classic computational problems such as execution of basic logical operations

[14], spatial logic and process algebra [15]. A typical example of its problem-solving capacity is its ability to find the shortest path to a target destination through a maze. [16]. Adamatzky and Jones reported that Physarum goo was able to optimise routes between ten points on a hypothetical map. They placed nutrients on each of the points to attract the goo and salt crystals on areas where it should not cross. Physarum goo's solution was comparable to that of a conventional computer programmed to solve the same problem [17]. For an overview of computing devices built with Physarum goo please refer to [1].

4 SONIFICATION OF PHYSARUM GOO

The movement of intra-cellular components inside Physarum goo's body and its protoplasmic tubes, and migration of the slime over a substrate, produce electricity that can be measured with electrodes strategically placed on the surface where the slime is being cultured. Figure 4 shows a typical setup using an electronic circuit that converts the electrodes' reading into digital format for further processing.

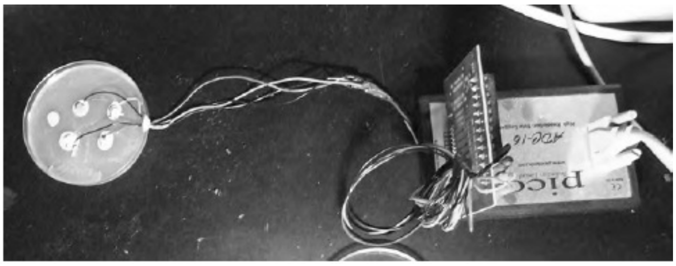


Figure 4: The electrical activity of Physarum goo can be recorded with electrodes. (Courtesy of Andy Adamatzky, University of the West of England.)

A recent study reported patterns of electrical activity that uniquely characterise Physarum goo's spatial dynamics and physiological states [18]. Different measurements of electrical potentials, or voltages, indicated when the plasmodium occupied and left specific sites. They also indicated when the organism entered into dormant stage.

In a previous collaboration with Adamatzky and Jones [19], we placed eight electrodes coated in non-nutrient agar gel in a petri dish, separated by a non-conductive material. To begin with, we placed Physarum goo on only one of the electrodes. Then, we placed an oat flake on top of each agar blob to act as attractors, or nutrients. During the course of one week, the goo colonized the other seven electrodes.

The electrodes measured the activity of the plasmodium, and the voltage readings were relayed to a synthesiser that translated them into sounds. Figure 5 plots the electrical activity of the first four electrodes with oat flakes over the week. For a discussion on the meaning of these graphs in terms of spatial dynamics and physiological states please refer to [18, 19].

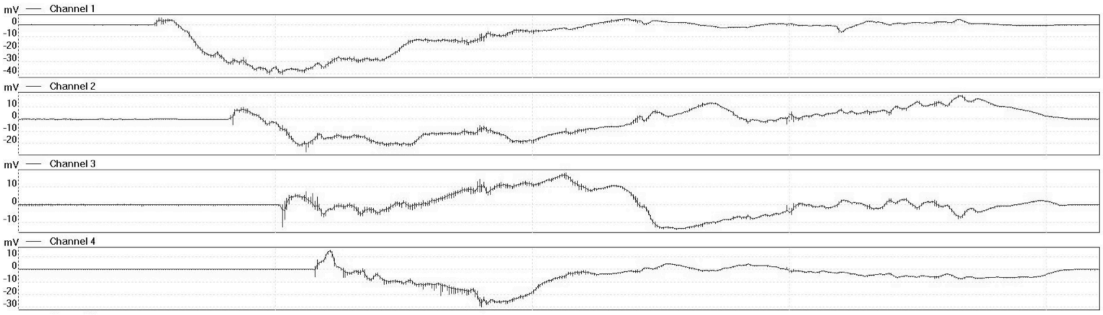


Figure 5: The electrical activity of Physarum goo measured by four electrodes over one week. The staggered beginning of electrical activity represents the goo reaching one electrode after another. (First published in http://arxiv.org/abs/1012.1809)

The goo spread amongst electrodes rather slowly. We recorded the voltages every second during the course of one week. This generated an excessively huge amount data, which would result in a rather long and slowly changing monotonous sound. To circumvent this problem, we devised a method to compress the data in order to render them suitable to produce a few minutes of sound. The compressed data still represented the overall behavior of the goo.

We rendered Physarum goo’s voltages into sounds by means of an additive granular synthesizer. Granular synthesis works by generating a rapid succession of short sound bursts referred to as *sound granules*. A sound granule is normally composed of partials, each of which is a sine wave generated by an oscillator. An oscillator needs two parameters to produce a sine wave: frequency and amplitude; phase information is sometimes needed, but we did not use phase information on this occasion.

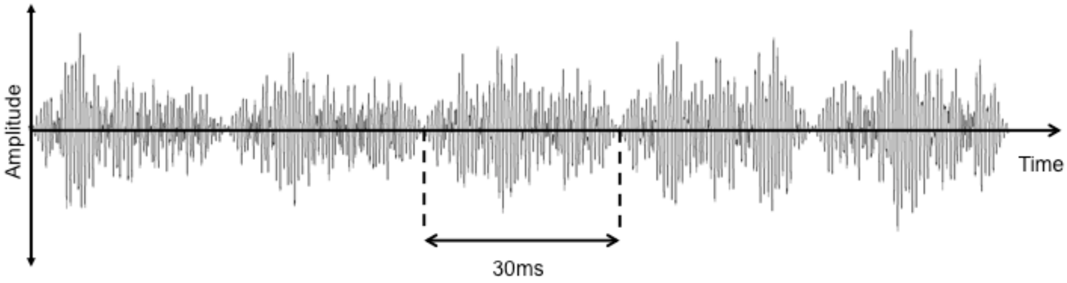


Figure 6: A sound formed of five granules lasting 30 ms each.

In our case, each granule is composed of seven partials: each of the seven electrodes generated data for a different oscillator. Voltage measurements from each electrode were converted into frequency and amplitude values for oscillators. In standard granular synthesis the duration of each granule is typically set in terms of tens of milliseconds. We produced sounds with granules ranging from 30 to 150 ms. Indeed, depending on how the synthesiser is programmed, such value can change dynamically as the sound is being synthesised. Figure 6 plots a 150ms long sound containing five granules of 30 ms each.

5 COMPUTER SIMULATION: JEFF JONES' PIXIEDUST

Jeff Jone's PixieDust is a multi-agent system simulator, which is able to simulate the behavior of Physarum goo [21]. This realistic simulator is useful because it is not tied to the time that the real Physarum goo would take to develop. PixieDust can generate reasonably credible data in a matter of minutes rather than weeks. In addition to being able to simulate behavior, PixieDust can also simulate the electrical activity measured by electrodes *in vitro*.

The interface of the simulator provides a window that displays a pre-loaded image representing the environment in which a population of agents representing Physarum goo would evolve (Figure 7). The image should normally be greyscale with 256 shades of grey, ranging from black to white. Each scale of grey has a meaning for the simulator; for example, completely white pixels are regarded as nutrients. The agents are released into this environment and act according to simulation parameters specified on a control panel.

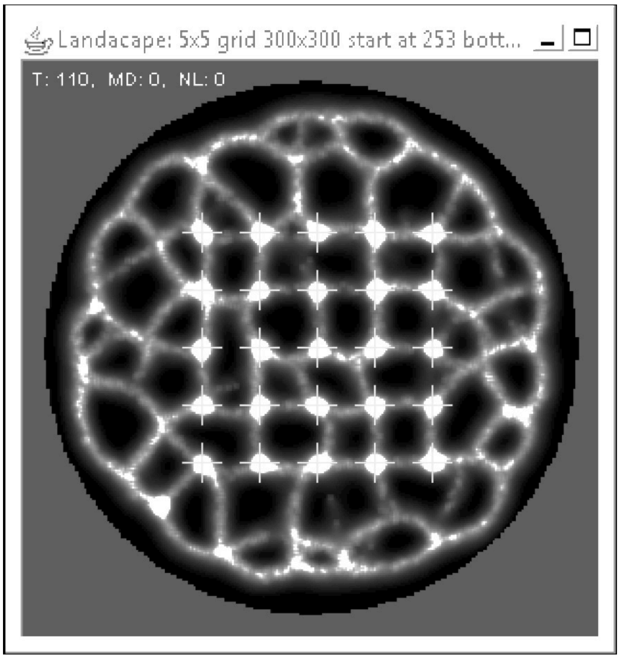
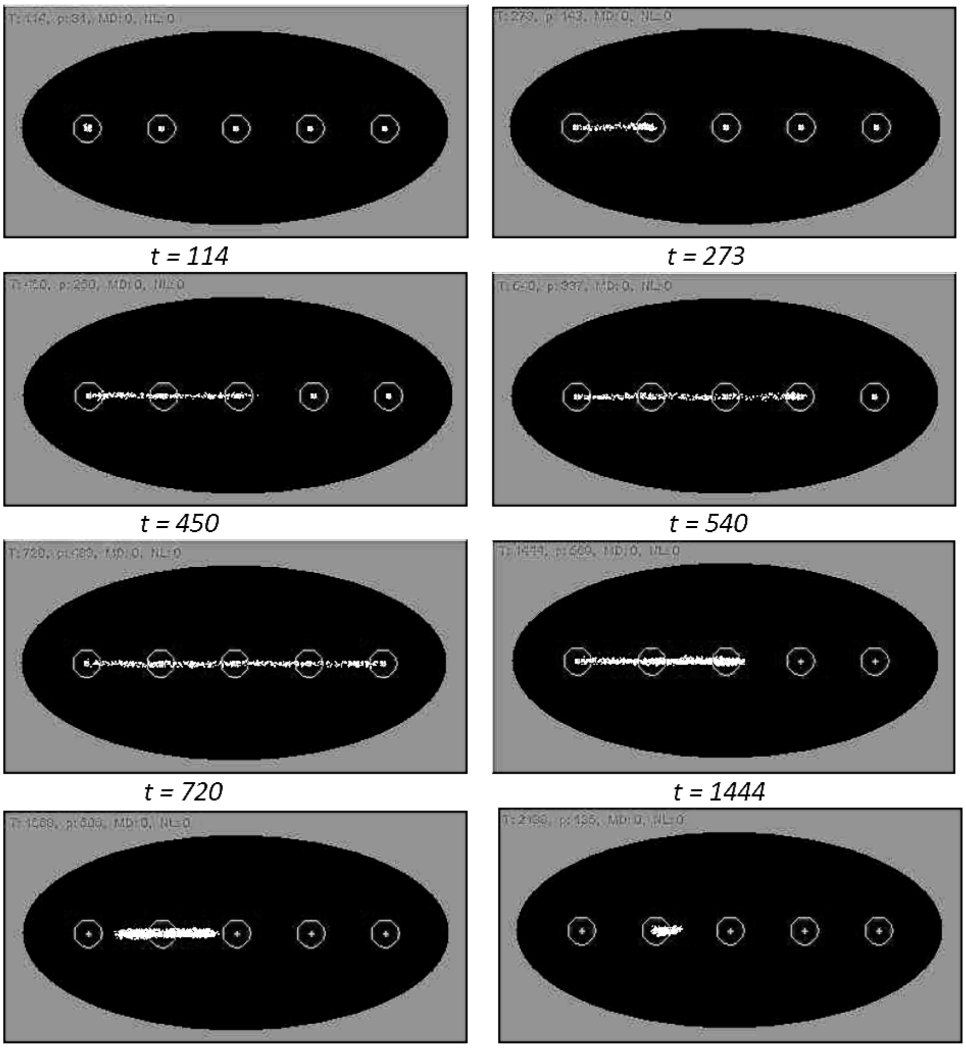


Figure 7: PixieDust's interface with a pre-loaded image representing the environment for the simulated Physarum goo. The crosses with a blob in the middle represent electrodes.



t = 114

t = 273

t = 450

t = 540

t = 720

t = 1444

t = 1568

t = 2138

Figure 8: Snapshots of PixieDust simulation, where *t* represents the simulation time steps the snapshots were taken.

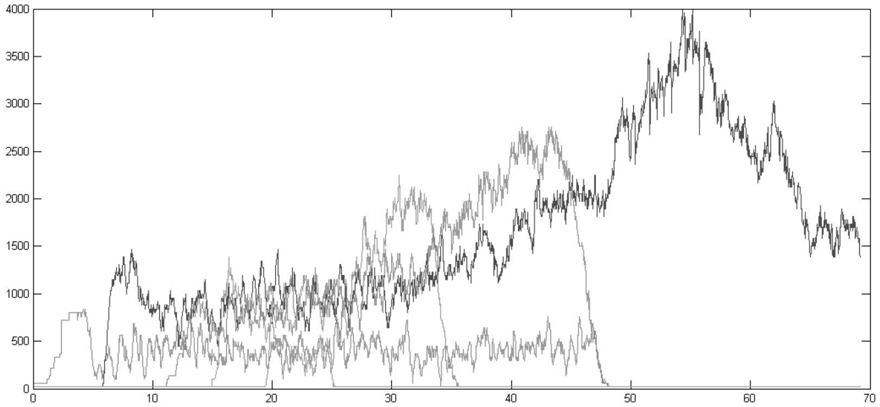


Figure 9: Plotting of the values of each electrode during the simulation over 70 secs. Each line corresponds to one of the five electrodes.

A detailed explanation of the simulation parameters is beyond the scope of this paper. It suffices to say that a simple stimuli-response algorithm governs the behavior of the agents. An agent occupying a certain location of the environment will move according to environmental stimuli, which determine the direction of its course. In reality, Physarum goo is indeed sensitive to the gradient of concentration of nutrients, thus in the simulation agents are attracted to locations with pixels representing food. The simulator allows for placement of virtual electrodes anywhere on the image representation of the environment, to collect virtual electrical potentials.

In order to probe PixieDust, we simulated an experiment described in [18]. We set the environment with five areas holding nutrients. An electrode was placed on each area. A cross in a circle represents an electrode. The radius of the circle defines the electrode's sensitivity. The value of an electrode at a given instant is given by the amount of agents inside the circle. Agents were initialized on the leftmost area, with electrode number one. As in the experiment with the real Physarum goo the agents in our simulation went to colonize the other areas. After all the all areas were eventually colonized, food was withdrawn and the agents gradually gathered around area with electrode number two (Figure 8). Figure 9 plots the values of each electrode during the simulation, showing that it is consistent with the paper. One can observe an overall oscillatory phenomenon, which is characteristic of the flow of cytoplasm inside the plasmodium. In conclusion, we were satisfied that the simulation provides data that is realistic enough for the purposes of this project.

6 PHYSARUM MUSIC SEQUENCER

The ability of Physarum goo to solve mazes, find shortest paths, and so on, informed the design of a music sequencer. More specifically, the sequencer takes advantage of the capacity of Physarum goo to develop networks linking sources of nutrients.

A music sequencer is a device that triggers elements from a programmed sequence of sounds, music notes and/or musical operations on a step-by-step basis at regular time intervals. The *Physarum Music Sequencer* was implemented as follows: eight electrodes were placed on the environment forming a circle. Each electrode is associated to an event of a sequence of events; e.g., a sound sample. The system reads the level of activation of the electrodes clockwise in sequence according to the tick of an imaginary metronome. If an electrode's activity is above a specified threshold, then the respective event associated to the electrode is triggered; e.g., a sound is played. Otherwise, nothing is triggered. After the activity of the eighth electrode of the circle is read, the system proceeds to read the activity of the first again, and so on, forming a continuous loop. Therefore, the eight electrodes yield a sequence of eight musical events; for example, eight sounds played in loop, some of which might be skipped when the electrode's activity is below the triggering threshold. An example is shown in Figure 10, where all electrodes hold nutrients at the beginning of the simulation. Physarum goo forms a circle binding all eight nodes. Then, when electrodes one, four, six and eight are void of nutrients, only electrodes two, three, five and seven remain populated with agents, yielding in a rhythmic sequence.

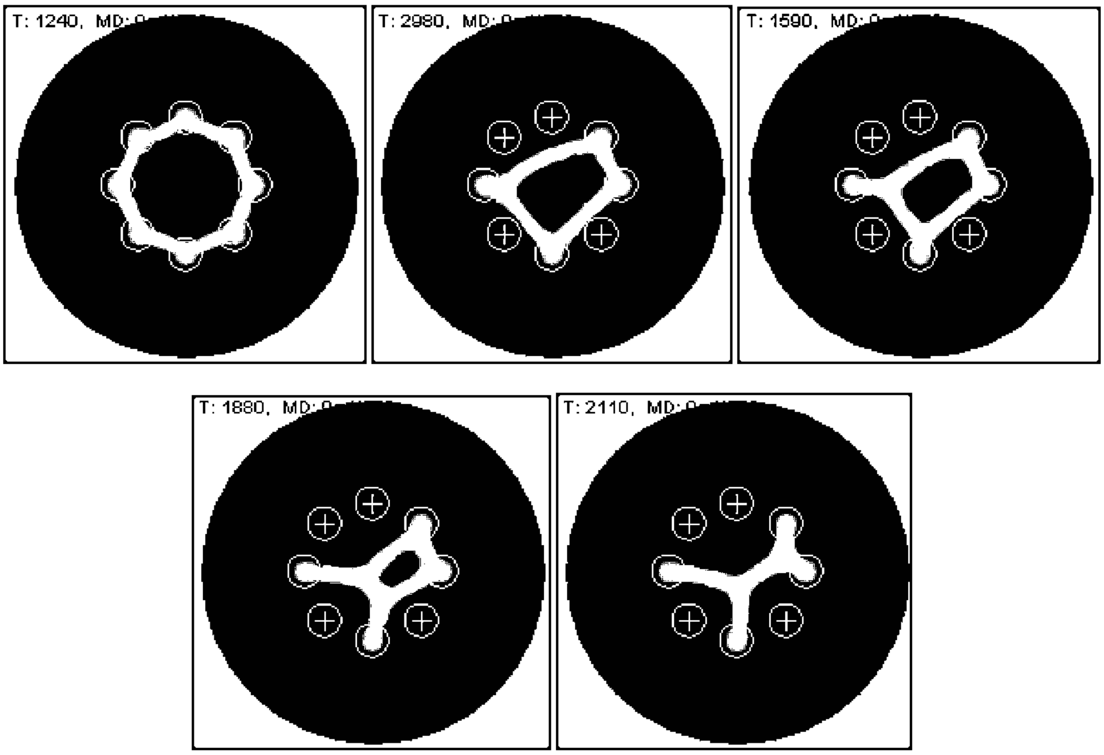


Figure 10: Physarum Music Sequencer.

7 THE COMPOSITION

For the composition *Die Lebensfreude*, six copies of the sequencer were arranged on the environment to form a shape resembling a flower with six petals. The composition is for flute, clarinet, piano, violin and violoncello, and six channels of electronic sounds. Therefore, each sequencer of the flower arrangement generated a sequence of notes for a different instrument; the piano, however, required two petals, one stream of musical notes for each hand. Each flower also generated information for a synthesizer to produce a channel of electronic sounds using the sonification method introduced in section 4. The flower's stem also contains nodes with nutrients for the simulation but it does not generate music.

Each cycle of the sequencer generated eight notes for each measure of the music at four beats per measure; i.e., time signature of 4/4. The tempo was fixed to120 beats per minute.

The simulation worked as follows: a few agents representing Physarum goo were placed at the bottom of the flower's steam. Then, gradually the agents evolved through the stem towards the petals.

The note sequences allocated to the petals are shown in Figure 11. As the electrodes on the stem do not produce any musical information, the generative music process per se started to take place when the agents reached the petals (Figure 12). Figure 13 shows an excerpt of the score generated by the system. Each measure corresponds to one cycle of the sequencer. For instance, the first measure of the flute part shows two notes, the first and the seventh notes of the flute's scale shown in Figure 11.

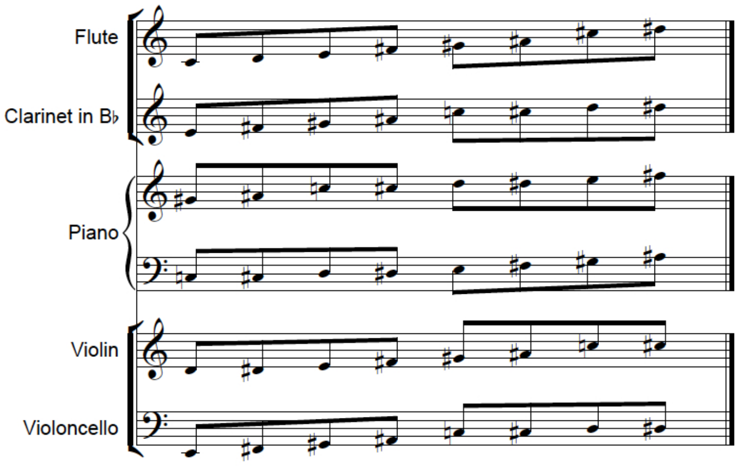


Figure 11: Each petal of the flower corresponds to a sequencer, which generates notes for one instrument.

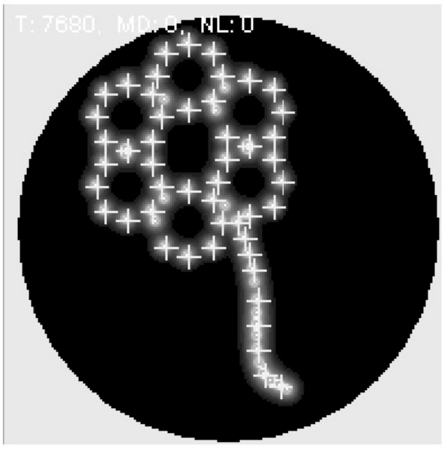


Figure 12: The system generates the musical score as the agents colonises the sites with the electrodes, represented by crosses, and consume the flower.

This means that at those particular moments only the first and the seventh electrode of the flute petal held enough agent activity to trigger the notes. The other notes remained silent. Another example, the second measure of the violin part has the second, fourth, sixth, seventh and eighth notes of the respective scale shown in Figure 11. In this case, the agents were active on the second, fourth, sixth, seventh and eighth electrodes of the violin petal, respectively. As the simulation runs, Physarum goo gradually consumes the flower (Figure 14). The composition terminates when the flower is totally consumed.

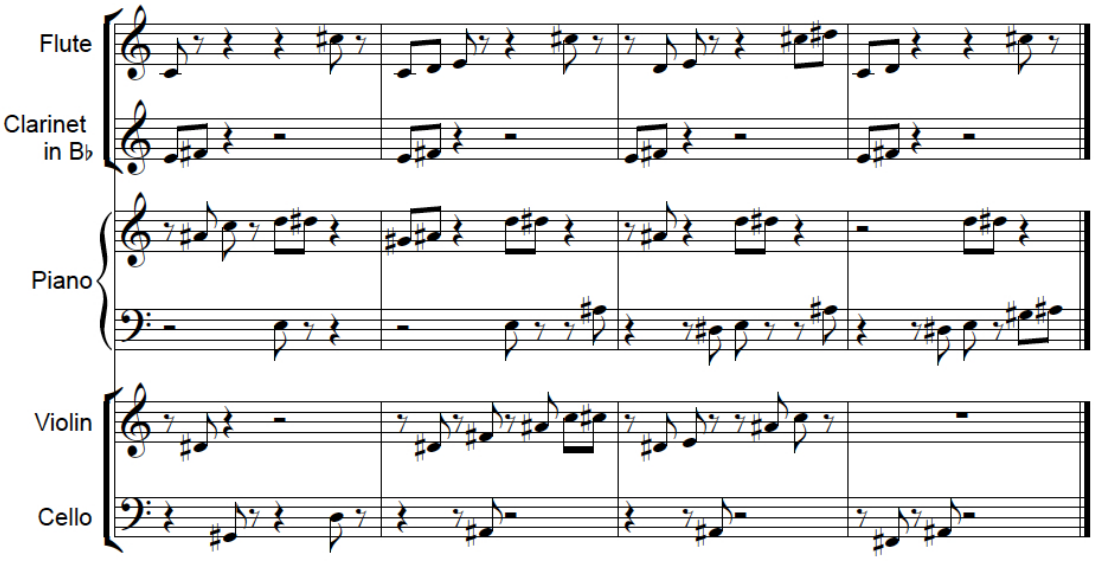


Figure 13: An excerpt of a musical score generated by the system from the sequences of notes shown in Figure 11.

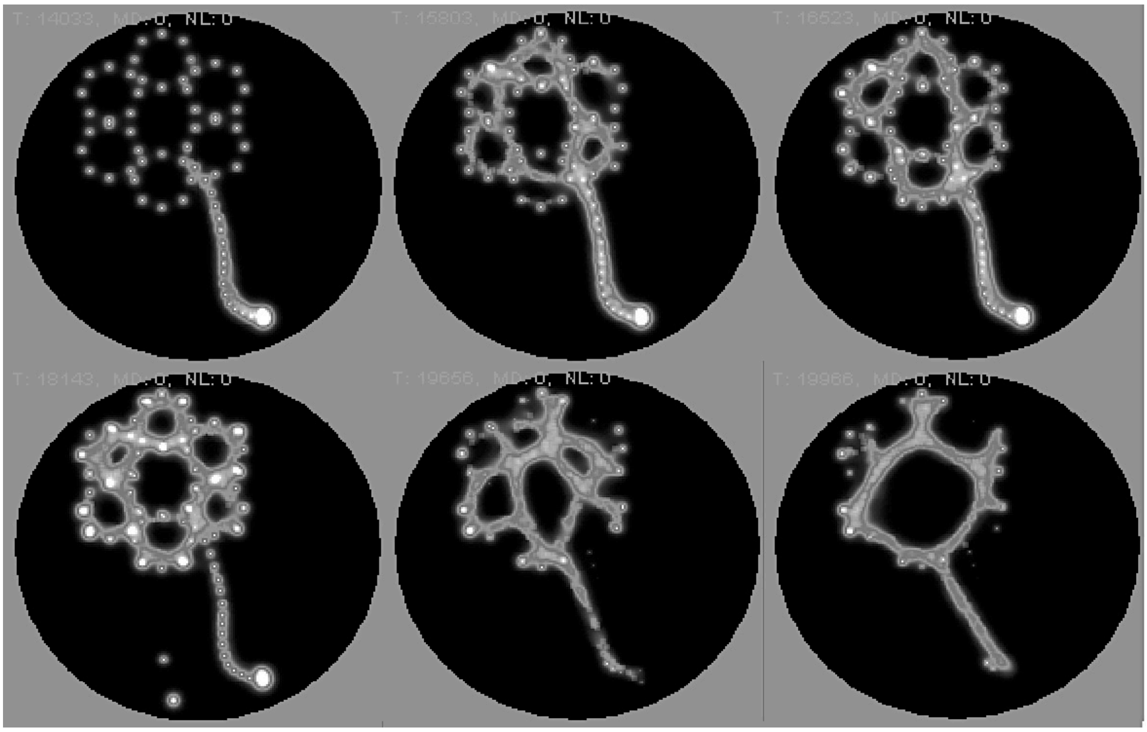


Figure 14: In the simulation, the flower is gradually eaten by Physarum goo.

8 FINAL REMARKS

Earlier in this paper I briefly discussed approaches to using computers in composition. Clearly, in the composition *Die Lebensfreude* I adopted the generative utilitarian approach: I considered the output from the system as raw materials for my piece. However, the changes I made to the materials are not global changes, but local ones. The overall form of the music generated by the simulation was not changed. The rhythmic structure remained largely the same. What I changed most were the pitches of the notes. Also, I added articulation to the materials; that is, I specified the way in which the notes are to be played, their loudness, and so on. For instance, sometimes the strings of the violins a plucked rather bowed, and the flute makes key slaps noise rather than a clean pitch, and so on. Moreover, I occasionally altered the speed of the music; it occasionally went faster or slower than the original120 beats per minute. An excerpt of the actual score is shown in Figure 15. The flutist is instructed to produce the sounds of key slaps whereas the pianist is asked to pluck the strings of the piano rather than play with the keyboard. The violinist and cellist are asked to play tremolando behind the bridge of the instrument.

In a recent book chapter [20], I offered a discussion on the role of the computer in my compositional practice, which is relevant to the work presented in this paper. In a nutshell, the role of the computer in my compositions has oscillated between two extremes: on the one hand, I have simply assumed the authorship of compositions that were entirely generated by a computer, albeit programmed to follow my instructions. On the other hand, I have composed with pencil on stave paper, using the computer only to typeset the final score. I shall argue that both approaches to composition are not incompatible, but manifestations of creative processes that are becoming progressively more polarized for me due to increasingly sophisticated technology. One side of me is very methodical and objective, keen to use automatically generated music, computers systems, formalisms, models and so on. Conversely, another side of me is more intuitive, emotional and metaphorical. Each side has it own agenda, so to speak, but they are not unrestrained, in the sense that they tend to inhibit each other: the more I attempt to swing to the objective side, the stronger is the intuitive force that pulls me to the opposite side. And vice-versa. I believe that the further my objective side pushes me to approach music according to its agenda, the stronger the pull of my intuitive side to approach it differently.

Hence, computer technology is of foremost importance for my métier, because it allows me to stretch my objective musical side far beyond my ability to do so by hand, prompting my intuitive side to counteract accordingly. However, I feel that the experience I gained from composing *Die Lebensfreude* with *Physarum polycephalum* has somehow bent this dichotomy. Yes, I am still using a machine. But it is a different kind of machine; it is a living organism. Indeed, the piece was generated by a computer model, but this might certainly not be the case in the future. As we move on to work with living matter, essentially we will be harnessing the intelligence of such organisms to compose music with. Undoubtedly, new forms of music making will emerge from Unconventional Computing. *Die Lebensfreude* is only a glimpse of what is to come.

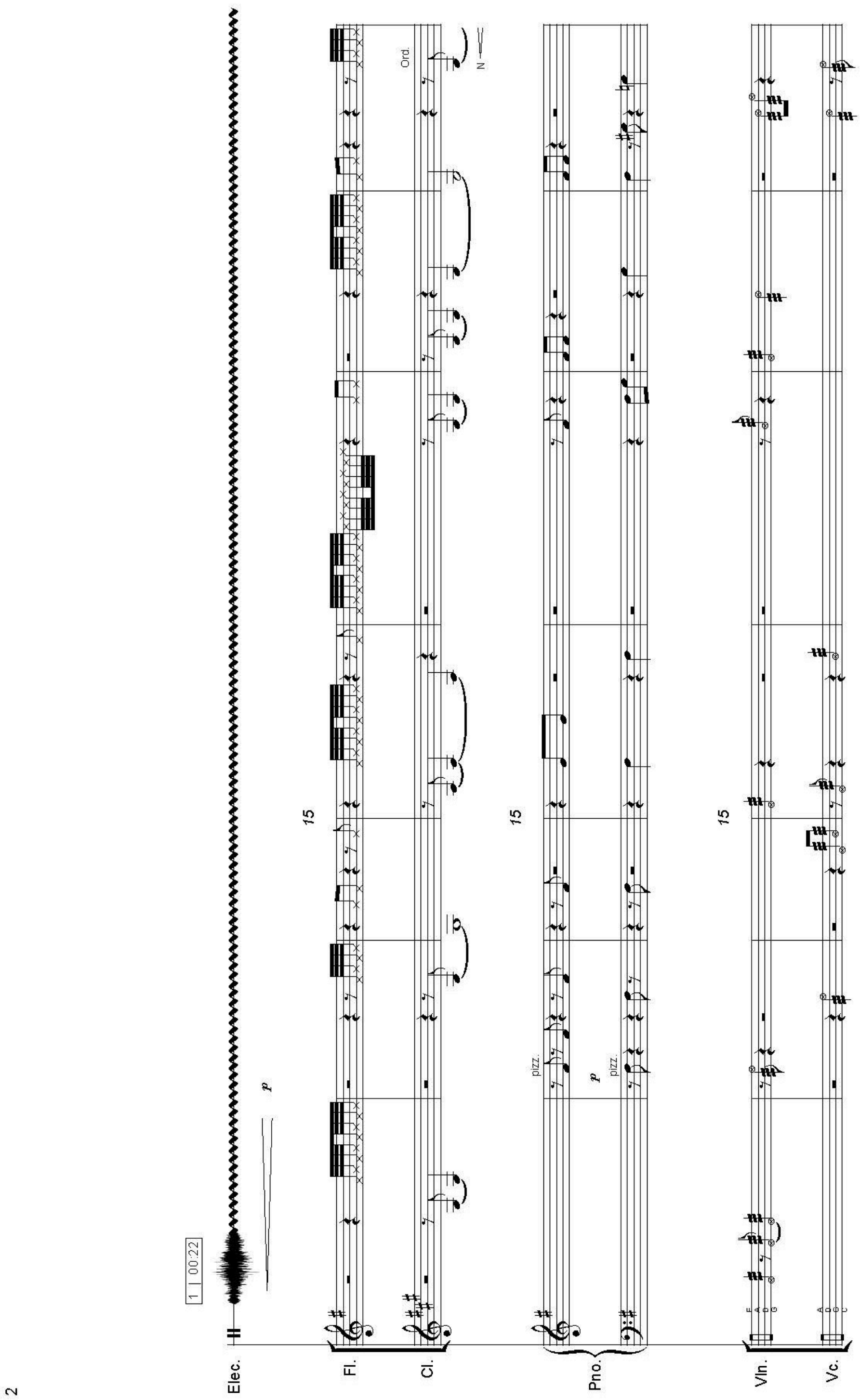


Figure 15: A sample page of the score for *Die Lebensfreude*.

I would like to thank Jeff Jones and Andy Adamatzky for their valuable contribution to this work. The simulation for the composition was developed by Dimitri Papadimitriou using Jone’s software [22] as part of his placement at ICCMR. A movie of the premiere, which took place at Cascais Cultural Centre, Portugal, in June 2012, by Sond’Ar-te Electric Ensemble, conducted by Guillaume Bourgogne, is available:
http://vimeo.com/channels/miranda/55536077

References

[1] Adamatzky A. (2010). *Physarum Machines*. Singapore: World Scientific.

[2] Miranda, E. R. (2001). *Composing Music with Computers*. Oxford: Focal Press.

[3] Calude, C., Casti, J. and Dinneen, M. J. (1998). *Unconventional Models of Computation*. Heidelberg: Springer-Verlag.

[4] Miranda, E. R., McAlpine, K. and Hoggar, S. (1999). “Making Music with Algorithms: A Case Study”, *Computer Music Journal* 23(2):19-30.

[5] Miranda, E. R. (2004). “At the Crossroads of Evolutionary Computation and Music: Self-Programming Synthesizers, Swarm Orchestras and the Origins of Melody”, *Evolutionary Computation* 12(2):137-158.

[6] Miranda, E. R. (2002). “Emergent Sound Repertoires in Virtual Societies”, *Computer Music Journal* 26(2):77-90.

[7] Jacobs, J. P. and Regia, J. (2011). “Evolving Musical Counterpoint: The Chronopoint Musical Evolution System”, *Proceedings of the First International Workshop on Evolutionary Music - 2011 IEEE Congress on Evolutionary Computation*, New Orleans, USA.

[8] Dabby, D. S. (1996). "Musical variations from a chaotic mapping", *Chaos*, 6:95–107.

[9] Dodge, C. (1988). “Profile: A Musical Fractal”, *Computer Music Journal*, 12(3):10-14.

[10] Miranda, E. R. and Kirke, A. (2010). “Game of Life Music”. In A. Adamatzky (Ed.), *Game of Life Cellular Automata*. London: Springer-Verlag, pp. 489-501.

[11] Miranda, E. R. (1995). “Granular synthesis of sounds by means of a cellular automaton”. *Leonardo* 28:297–300.

[12] Miranda, E. R., Bull, L., Gueguen, F. and Uroukov, I. S. (2009). “Computer music meets unconventional computing: towards sound synthesis with in vitro neuronal network”. *Computer Music Journal*, 33(1):9–18.

[13] Adamatzky, A. and Teuscher, C. (2006). *From Utopian to Genuine Unconventional Computers*. Frome: Luniver Press.

[14] Tsuda, S., Aono, M. and Gunji, Y. P. (2004). “Robust and emergent Physarum logical-computing”. *Biosystems* 73:45–55.

[15] Schumann, A. and Adamatzky, A. (2009). “Physarum spatial logic”. *Proceedings 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Timisoara, Romania.

[16] Nakagaki, T. (2000). “Intelligence: Maze-solving by an amoeboid organism”, *Nature*, 407:470.

[18] Adamatzky, A. and Jones, J. (2010). “On electrical correlates of *Physarum polycephalum* spatial activity: Can we see Physarum Machine in the dark?”, *Biophysics Reviews Letters* 6:29-57.

[19] Miranda, E. R., Adamatzky, A. and Jones, J. (2011). “Sound Synthesis with Slime Mould of Physarum Polycephalum”, *Journal of Bionic Engineering* 8:107-113.

[20] Miranda, E. R. (2012). “On Computer-aided Composition, Musical Creativity and Brain Asymmetry”. In D. Collins (Ed.), *The Act of Musical Composition: Studies in the Creative Process*, pp. 215-232. Farnham: Ashgate.

[21] Jones, J. (2010). “The Emergence and Dynamical Evolution of ComplexTransport Networks from Simple Low-Level Behaviours”, *International Journal of Unconventional Computing* 6(2):125-144.

[22] Jeff Jone’s website: http://uncomp.uwe.ac.uk/jeff/ (Accessed on 05/03/2013).