The Mégra System - Small Data Music Composition and Live Coding Performance

Niklas Reppel
Eurecat Barcelona
nik@parkellipsen.de

 ICLC 2020, February 5-7, 2020, University of Limerick ,Limerick, Ireland

Abstract

This article describes the Mégra music system, a code-based, stochastic music system that can be used in a live performance context, as well as for longer compositions. Mégra relies on Probabilistic Finite Automata (PFA) as its fundamental data structure. A case is made for the use of PFAs as a data model that can not only be trained (in the sense of machine learning), but also be interacted with on the basis of predefined operations and, as a side effect, enables one to creatively use the imperfections that occur when using very small data sets to infer musical sequence generators with the help of machine learning methods.

Introduction

Mégra is a code-based system that can be used to create music in a live performance context, as well as in a composition context. It allows one to interactively infer or create musical sequence generators by using the live coding or exploratory coding method.

Its development started out from the idea of, as Fiebrink and Caramiaux (2018) put it, 'machine learning in which learning algorithms can be understood as a particular type of interface through which people can build model functions from data,' and the subsequent search for a suitable data model. Further criteria were the efficiency of the training methods so that they can be used in the context of real-time music creation, as well as a syntax that reflects the model well while being sufficiently intuitive for live coding.

The 'imperfections' which occur when using very small data sets (small enough to be observed and entered by humans) are a welcome side effect of the chosen approach.

Furthermore, the system attempts to include methods to manipulate the learned structures by means other than just choosing the input data. Instead, it aims to employ a model that is more semantically meaningful in comparison to other models currently common in machine learning and artificial intelligence, such as deep neural

networks. This semantic quality at a high level should allow for the definition of more meaningful operations and interactions inside the model, at a granular level. In that sense, the Mégra language presents a case study on how a machine learning method can be put into action within live coding practice.

Section 2 of this paper will evaluate this idea in the contexts of Algorithmic Composition and the current Big Data trend, or in opposition of the latter. Section 3 will evaluate the chosen data model and eventual alternatives. Section 4 will give a brief overview over the Mégra system, and how the model is used in the context of live coding. Section 5 will present some usage examples.

Small Data as a Creative Tool

The part of the machine learning world commonly associated with the Big Data buzzword seems to be dominated by huge data sets and huge hardware effort. While it is hard to deny that the results are impressive, they are still mostly beyond reach for people that are not part of a company or a research group, as neither the necessary hardware (like supercomputers or AI accelerators) nor the data sets are commonly available[1].

Furthermore, 'black-box' models that are commonly in use, like deep neural networks, barely have any semantic significance for humans (one might say they aren not *cognitively available*). The semantic meaninglessness of the data models makes a 'discourse with models' (Roberts and Wakefield 2018, p. 303), e.g. by applying meaningful operations to them, quite hard, apart from 'shaping model behavior through data' (Fiebrink and Caramiaux 2018). Manually modifying the internal structure of the 'black box' would not only be tedious, but also virtually impossible to do in a controlled manner.

Here is where the idea of Small Data comes into play. The term came up in different fields, most prominently in marketing (Lindström 2016) or in e-Health systems research (Estrin 2014).

There does not seem to be a final consensus on what constitutes Small Data. The idea is usually characterized in direct opposition to Big Data, for example by data sets that can be generated by a single person (Estrin 2014) or that are within the realm of human comprehension, leading to intuitive insight (Lindström 2016). Decentralization of the data sets also plays a role (Pollock 2013).

In this specific context of interactive music creation, the aspect of smallness also can be extended to direct feedback. While Big Data applications usually need, in addition to hardware, a lot of time to give accurate results, small data sets can be immediately evaluated and a structure inferred within an exploratory, real-time-oriented compositional approach, as is typical for live coding methods.

Furthermore, the idea of smallness can be extended to the model itself, where the observation of the model or rules inferred from the data can lead to further insight, as semantically meaningful models are preferred.

Using small data sets and real-time inference surely won't return 'perfect' (or perfectly predictable) results (e.g., in the sense of a stylistic imitation that can't be distinguished from an original). Given the current state of technology, the representational capacity of the methods proposed in the following are behind of what is currently achieved with deep learning methods and the like. In the context of artistic production, though, this isn't the only criterion. In fact, it might not be all that significant as long as the results are inspiring and artistically valid, with the methods being available to anybody with access to regular hardware.

In some sense the Small Data idea is also reminiscent of the early days of computer composition, when neither the amount of data nor the hardware capacity was anywhere near today's level, and more

[1] It should be mentioned that in recent times some effort has been made to make machine learning technology more widely available, with projects like the Julia language (Innes 2018), or, in the context of music (especially synthesis and instrument creation, as opposed to structure generation), the Nsynth dataset and GANSynth (Engel et al. 2019). Their usefulness in the context of live coding has yet to be evaluated.

domain-specific knowledge may have been needed to achieve valid results. Today's technical world allows us to re-create some of this in real time.

Thus, the idea of Small Data in the given context could be seen as less of a black box approach, but a more human, democratic use of machine learning methods (Pollock 2013), not with the goal of giving accurate predictions or categorizations of the world around us, but rather for artistic inspiration, by means of discourse with the model itself.

Probabilistic Finite Automata

Using Markov Chains of different orders for musical sequence generation is a well-known method in algorithmic composition (Nierhaus 2009). Common Music, for example, provides methods to create sequence generators based on those (Taube 2014).

Probabilistic Finite Automata (PFA) (Ron, Singer and Tishby 1996) are a representation of Variable Order Markov Chains, which combine the predictive power of fixed higher-order Markov chains (where needed) with a more compact memory footprint.

Mathematically, PFAs are described by a 5-tuple $Q, \Sigma, \Gamma, \Upsilon, \Pi$ and a memory length N, where:

- Q is a finite set of states, the states being labeled over $\Sigma^n, n \leq N$;
- Σ is a finite alphabet;
- Γ is the transition function, determining the next state given a current state and an emitted symbol;
- Υ is the next symbol probability function, determining the probability of a symbol

The PFA model isn't necessarily interesting for its novelty, but for its versatility, and in this case, the ability to support a semantically meaningful dialog. The semantic quality (i.e. the capacity for transmitting meaning) of the model becomes clear if you think of the elements of the alphabetas musical events. Thus, the states, their labels and the transitions between them can easily be read in natural language ("after four repetitions of a bassdrum follows a snare with 50

Probabilistic Finite Automata can be used as sequence generators, whether they are inferred from user-provided rules or trained from given sequences of observations. The training and inference algorithms are efficient enough to be used in the context of real-time, on-the-fly composition (as the examples in the following sections will show).

Furthermore, it is fairly straightforward to interact with and manipulate the learned or inferred structures to create variation, either by adding rules or by manipulating the inferred structures directly through predefined operations. Thus, PFAs are a good approximation of the Small Data idea and present a data model that is trainable in the machine learning sense while also allowing for discourse with the model through live interaction.

Especially with limited data sets, the model is somewhat intuitive. Smaller sequence generators (that might nonetheless produce interesting results) can be even written by hand or drawn onto a sheet of paper (Fig. 1). This intuitive accessibility corresponds well to the idea of using Small Data for artistic inspiration.

Findings by David Huron might give hints regarding the usefulness and limitations of the PFA model in relationship to human cognition. As Huron writes:

> In describing conditional probabilities, two concerns are the contextual distance and contextual size. Some states are influenced only by their immediate neighbors (i.e., small contextual distance). Other states are influenced only by states that are far away in space or time (i.e., large contextual distance). [...] The size of the context of probabilistic influence is sometimes also called the probabilistic order. [...]

> As we will see in later chapters, music exhibits a complete range of such dependencies. Most of the time, the principal constraints are of low probability order and involve a near context (e.g., one note influences the next note). But music also exhibits distinctive patterns of organization where distant contexts are more influential than near contexts and the probability order is quite large. (Huron 2006, p. 56)

In the context of live coding, the smaller contexts can be easily represented by the PFA model, as we will see later on, while, due to restrictions regarding processing power and time, the larger contexts are still in the hands of the performer.

Alternative Models

Among alternative models that might fit the Small Data idea, in that they are comparatively human-readable, one might be Augmented Transition Networks, as previously applied by David Cope (Nierhaus 2009), even though based on extensive musical corpus analysis rather than intuitive data entry in a live coding situation.

Generative Grammars or Probabilistic Generative Grammars (Nierhaus 2009) might also be considered, even though they might be more suited to offline sequence generation rather than real-time generation due to the way non-terminal symbols are handled.

The use of these models in the context of live coding needs further research to determine their practicality.

The Mégra System

In Mégra, the mathematical details are transparent to the user; only the details essential to interaction made it to the syntax. Also, the model can be visualized fairly easily, as seen in the code examples and their visualizations 1-4.

The system embodies the Small Data idea by providing a compact and semantically meaningful syntax both for creating sequence generators by hand, to gain a better understanding of the underlying model, as well as for inferring structures from tiny data sets, which can then again be turned into code, visualized, manipulated, and, of course, sonified.

Learning, Inferring and Extending Structures, Making Up Rules

The Mégra system allows for the creation of musical sequence generators on the fly in several ways. One way is to explicitly specify a set of transition rules (Listing 1).

```
;; Code Example I: a simple beat generator inferred
   from explicit rules
;; see visualisation in Figure 1
 (infer 'beat
     (events (x (sn)) (o (bd)) (- (hats))) ;; symbol-
        to-sound-event mapping,
                ;; x = snare, o = bassdrum, - = hats
     (rules
                ((x) - 1.0) ;; hats follows snare,
                   always
                ((o) - 1.0) ;; hats follow bassdrum,
                   always
                ((-) x 0.4) ;; after hats, either
                   have another snare,
                ((-) o 0.4) ;; ... another bassdrum,
                   ...
                ((-) - 0.2) ;; ... or, less
                   frequently, another hats.
                ((- - - -) o 1.0))) ;; after four
                   sequential hihat sounds, always
                   emit a bassdrum
```

Listing 1: Mégra Code Example I, a simple beat generator

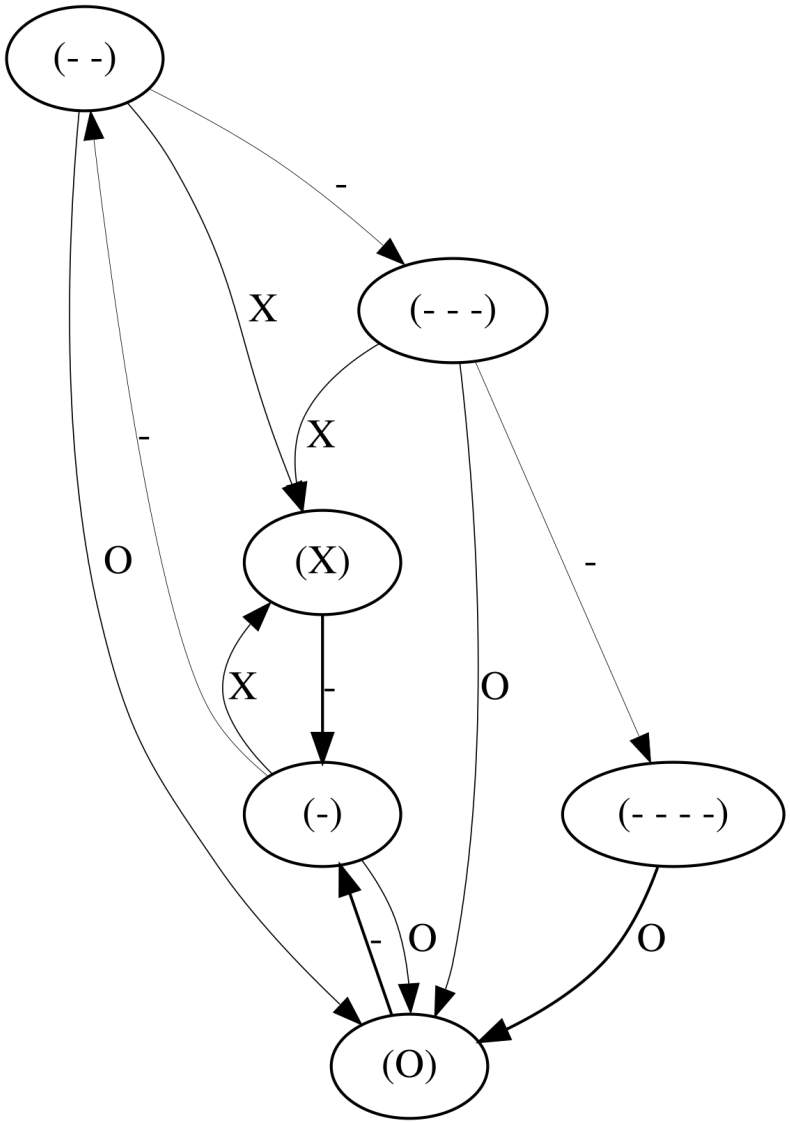

Figure 1: Graphical Representation of Code Example I

Another way is to train a sequence generator from an observed or made-up sequence (Listing 2). Once the sequences' structure has been inferred, it can be modified in different ways. A straightforward way would be to export the learned structure as code, such as in the first example, which is possible, but the amount of code might quickly grow too big to manually edit it.

```
;; Code Example II: a more complex beat generator
    learned from a pattern
;; see visualisation in Figure 2
(slearn 'beat
     (x (sn) o (bd) - (hats)) ;; symbol-to-sound-
         event mapping (shorter syntax)
       "x-o-x-o-x-o-x-o-x-o-x-o-x-o-x-o-x-
        o-x-o-x-o-x-ox-xo-x--ox-ox--xo-x-xo
        -xox-ox-ox-ox-xx-ox") ;; a sample string to
            learn from
```

Listing 2: Mégra Code Example II, Training Generator

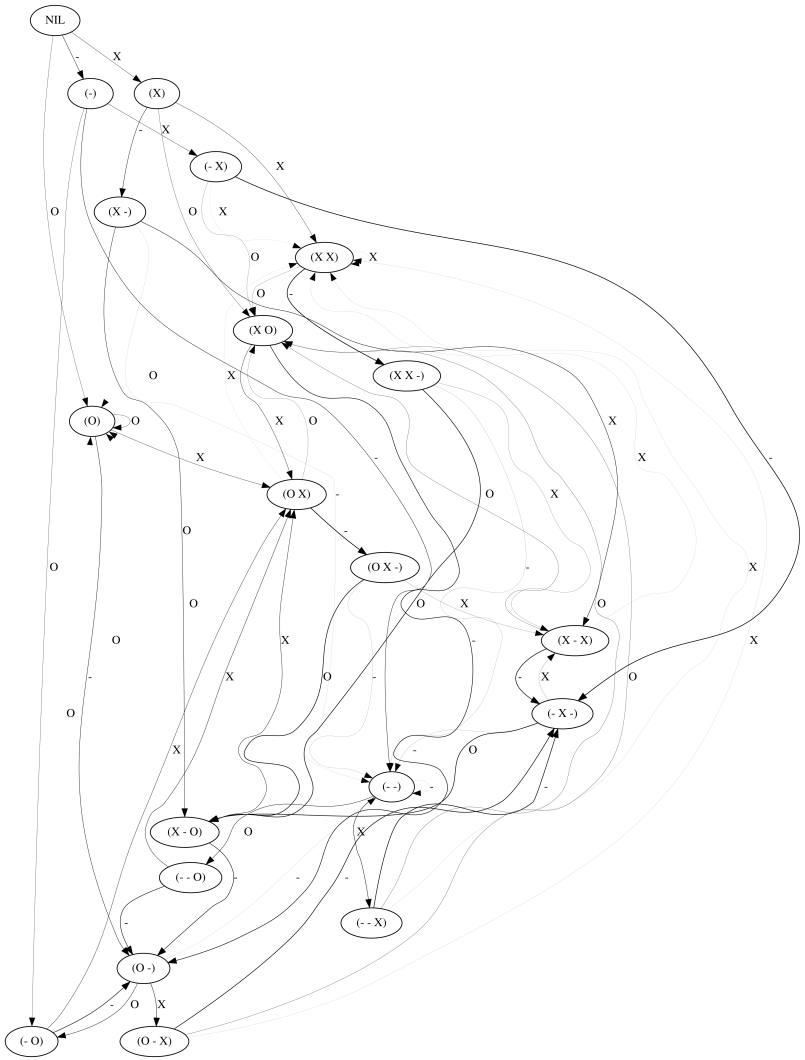

Figure 2: Graphical Representation of Code Example II

Thus, a more manageable interface with the model would be to manipulate the learned PFA by inserting additional states on the basis of predefined operations, and the actual history of emitted events. In that manner, it is also possible to start from a very simple structure (Listing 3 and Figure 3) and 'grow' the sequence generator successively by inserting nodes and edges following certain criteria.

```
;; Code Example III: a very simple starting point, a
     nucleus
;; see visualisation in Figure 6
(s 'the ()
  (nuc 'nucleus (saw 'a2 :dur 102 :atk 2 :rel 100 :
     lp-freq 1000)))
```

Listing 3: Mégra Code Example III, Training Generator

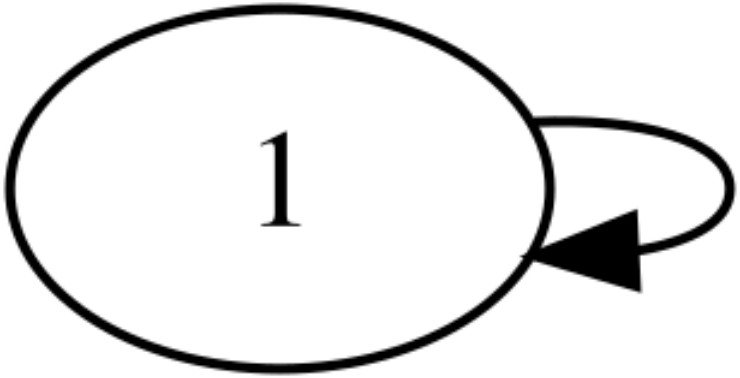

Figure 3: Graphical Representation of Code Example III

The growth operation, as described in the following (Listing 4 and Figure 4) is an example of this kind of interaction with the model. It

will spawn a new node based on the last one that has been evaluated, modify the parameters of the formerly emitted event(s) with an average variance of 0.3, and arrange the new edges in a way that small loops of three musical events will emerge.

```
;; Code Example IV: growth operation
;; successively extend - see visualisation after
    extension in Figure 4

(grow 'nucleus :variance 0.3 :method 'triloop)
```

Listing 4: Mégra Code Example IV - Growth operation

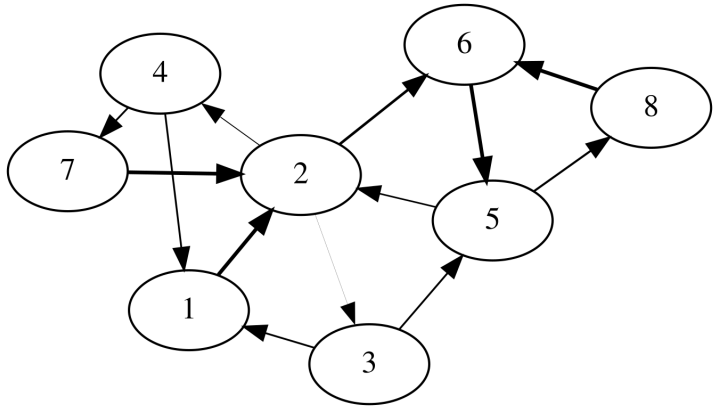

Figure 4: Graphical Representation of Code Example IV after growth iterations

The growth method above can again be automated, e.g. by a simple life-modeling algorithm. This gives each node a certain lifespan after which they perish and spawns new nodes after a specified amount of evaluations in the way described above, tied to the availability of a predefined amount of "resources." This way, the generated generators will stabilize (or perish) after a while, once the resources for further growth run out. If the context requires a more deterministic outcome, the Mégra system also allows for the creation of sequence generators from a more musically oriented description, e.g. from a layered loop (or pattern) syntax (Listing 5), which translates to the same PFA model and thus allows for using the same interactions later on.

3mm

```
 ;; Code Example V: layered loop syntax
(s 'beat ()
   (cyc 'layer2 "hats ~ ~ hats ~ ~ ~ hats hats") ;;
      loop 2
   (cyc 'layer1 "bd ~ ~ bd sn ~ ~ casio:'hi")) ;;
      loop 1
```

Listing 5: Mégra Code Example V

Time Handling

The examples above rely on a fixed time spacing (no explicit time information given), but it is also possible to have explicit time control, with time values specified in milliseconds (Listing 6).

```
 ;; Code Example VI: Same as example 1, with some
    explicit time values
(infer 'beat
     (events (x (sn)) (o (bd)) (- (hats)))
     (rules
                ((x) - 1.0 100) ;; hats follows snare
                    , always, after 100ms
                ((o) - 1.0 100) ;; hats follows
                    bassdrum, always
                ((-) x 0.4) ;; after hats, either
                    have another snare,
```

```
                ((-) o 0.4) ;; \dots another bassdrum
                    , \dots
                ((-) - 0.2 50) ;; \dots or, less
                    frequently, another hats (50ms)
                ((- - - -) o 1.0))) ;; after four
                    sequential hihat sounds, always
                    emit a bassdrum
```

Listing 6: Mégra Code Example VI, a simple beat generator with some explicit time control

Pragmatic Interaction

```
 ;; Code Example VII: Event Streams
(s 'sawtooths () ;; <- This is an event sink.
   ;; ^
   ;; | Events flow in this direction ...
   ;; |
   (prob 30 (rev 0.2))  ;; Modifier! 30% chance to
       add some reverb.
   (nuc 'nucleus (saw 'a2 :dur 102 :atk 2 :rel 100 :
       lp-freq 1000)) ;; Source
```

Listing 7: Code Example VII: Pragmatic interaction by modifying the event stream.

Especially in an exploratory situation it is sometimes helpful to quickly change certain parameters or create some variation by modifying a parameter with a certain probability.

The Mégra system allows for those pragmatic modifications (pragmatic in the sense that they're not necessarily covered by the PFA model) by altering the event stream with certain operators that select and modify the parameters of the passing events (Listing 7), inspired by what is commonly called Reactive Programming, e.g. as described in Maier et al. (2010).

[2]Demo: `https://vimeo.com/321099751`

Technical Foundations

Mégra is built upon Common Lisp as a base language and utilizes SuperCollider for sound synthesis. The two communicate via Open Sound Control.

The Common Lisp language has been chosen for its expressive power and the syntactical freedom it provides. Furthermore, there are several powerful libraries for music creation available, most prominently Incudine (Latini 2019) and an ancient, but functional, version of Common Music (Taube and Finnendahl 2019).

Usage Examples

In the following, two specific usage examples and some more generic remarks about the usage of the Mégra system will be presented.

Creating a Pattern Syntax on the fly

The Mégra system can be used to create small pattern languages on the fly by associating sound events with symbols and entering a sequence as a string[2]. The possibilities can be explored by modifying the string and eventually adding new symbol/sound associations.

The resulting syntax might be somewhat similar to other approaches (think of Gibber (Roberts n.d.) or FoxDot (Kirkbride 2019)), allowing the application of previous pattern knowledge. But, as a non-deterministic sequence generator is inferred, the results might not be totally expected. Code Example II above (Fig. 3) also follows a similar idea.

Simple Language Sonification

Another possible use of this system is to enter a slightly larger, given data sequence and to associate its symbols with sounds, e.g. to sonify text snippets[3] from 'simple' language texts like Toki Pona (Lang 2014).

Performance and Composition

The Mégra system has been continuously used for composition[4] and live coding concerts[5] since its inception, which has been an important aspect in its development process. Especially through the use in a live situation, many inconveniences in syntax and handling have been exposed and successively improved, and it is currently approaching a somewhat stable state. 6 FUTURE OUTLOOK While further simplifying the syntax and increasing expressive possibilities is an ongoing project, a major future goal is inferring the PFA structures not only from code input, but also from audio input. That way, a sequence generator could be created simply by clapping a rhythm, or singing a melody. This has, as a first step, required some audio feature extraction, which has been done by creating Common Lisp bindings for the popular Aubio library. As a next step, a method to transfer the extracted features into the symbolic domain is needed so that they can be used in the same manner as the code-based input.

Conclusions

With Probabilistic Finite Automata, a trainable data model that still allows for versatile real-time interaction and thus, discourse with the model, has been identified and implemented.

From personal experience, engaging in an active discourse with the PFA model, training or inferring sequence generators, keeping or discarding the results and subsequently applying the mentioned operations to transform the results allows for the frequent discovery of non-obvious, yet interesting sequences and sound combinations.

My aim with this project was to engage the audience (and myself as a performer) by creating and exposing a semantically meaningful discourse with a Small Data model through sound and code.

It does need some practice to be uses effectively, and the pragmatic interaction (as described in Section 4.3) is still an important part, especially in the context of performances that require more rapidly-shifting dynamics, such as Algorave. On its own (without much pragmatic interaction), the presented system shines in types of music that unfold more slowly in time.

Nonetheless, having used it successfully in performances so far, the Mégra system continues to be extended and field-tested.

References

Estrin, D. (2014) Small data, where n=me. Communications of the ACM, 57(4), pp.32-34.

Engel, J., Agrawal, K.K., Chen, S., Gulrajani, I., Donahue, C. and Roberts, A. (2019) Gansynth: Adversarial neural audio synthesis. arXiv preprint arXiv:1902.08710.

Fiebrink, R. and Caramiaux, B. (2018) The Machine Learning Algorithm as Creative Musical Tool. In The Oxford Handbook of Algorithmic Music (pp. 181–208). Oxford University Press. `http://doi.org/10.1093/oxfordhb/9780190226992.013.23`

[3]Demo: `https://vimeo.com/321099989`

[4]Demo: `https://ellipsenpark.bandcamp.com/track/hayaoki-ii-raintech`

[5]Demo: `https://www.youtube.com/watch?v=IJPKeKZ6bv0` (w/ Turbulente on visuals)

Huron, D. (2006) Sweet Anticipation. Cambridge, MA, USA: The MIT Press.

Innes, M. (2018) Flux: Elegant machine learning with Julia. J. Open Source Software, 3(25), p.602.

Kirkbride, R. (2019) FoxDot: Live coding with Python. [online] FoxDot: Live coding with Python. Available at: https://foxdot.org/ [Accessed 7 Sep. 2019].

Lang, S. (2014) Toki Pona - The Language of Good. Tawhid.

Latini, T. (2019) Incudine. [online] incudine.sourceforge.net. Available at: http://incudine.sourceforge.net [Accessed 7 Sep. 2019].

Lindström, M. (2016) Small Data. St. Martin's Press.

Maier, I., Rompf, T. and Odersky, M. (2010) Deprecating the observer pattern.

Nierhaus, G. (2009) Algorithmic Composition. Dordrecht: Springer.

Pollock, R. (2013) Forget big data, small data is the real revolution. [online] The Guardian. Available at: https://www.theguardian.com/news/datablog/2013/apr/25/forget-big-data-small-data-revolution [Accessed 8 Sep. 2019].

Reppel, N (2019) https://github.com/the-drunk-coder/megra. [online] Mégra. Available at: https://github.com/the-drunk-coder/megra [Accessed 7 Sep. 2019].

Roberts, C. (n.d.) [online] Gibber.cc. Available at: https://gibber.cc/ [Accessed 7 Sep. 2019].

Ron, D., Singer, Y. and Tishby, N. (1996) The power of amnesia: Learning probabilistic automata with variable memory length. Machine learning, 25(2-3), pp.117-149.

Taube, R. and Finnendahl, O. (2019) Common Music 2.12. [online] GitHub. Available at: https://github.com/ormf/cm [Accessed 7 Sep. 2019].

Taube, R. (2014) Common Music 3. [online] commonmusic.sourceforge.net. Available at: http://commonmusic.sourceforge.net/cm/res/doc/cm.html#markov-analyze [Accessed 11 Sep. 2019].

Roberts, C. and Wakefield, G. (2018) Tensions and Techniques in Live Coding Performance. In Oxford Handbook of Algorithmic Music (pp. 293–317). Oxford University Press. http://doi.org/10.5281/zenodo.1193540