Algorithmic Pattern

Alex McLean
Research Institute for the History of Science and Technology
Deutsches Museum, Munich
alex@slab.org

ABSTRACT

This paper brings together two main perspectives on algorithmic pattern. First, the writing of musical patterns in live coding performance, and second, the weaving of patterns in textiles. In both cases, algorithmic pattern is an interface between the human and the outcome, where small changes have far-reaching impact on the results.

By bringing contemporary live coding and ancient textile approaches together, we reach a common view of pattern as algorithmic movement (e.g. looping, shifting, reflecting, interfering) in the making of things. This works beyond the usual definition of pattern used in musical interfaces, of mere repeating sequences. We conclude by considering the place of algorithmic pattern in a wider activity of making.

Author Keywords

Pattern, TidalCycles, Algorithmic Music, Textiles, Live Coding, Algorave

CCS Concepts

•Applied computing → Sound and music computing; Performing arts; Media arts; •Software and its engineering → *Functional languages;*

1. DEFINING ALGORITHMIC PATTERN

This paper explores the world of algorithmic pattern, and the ways in which it offers an interface to the computer musician. To introduce this topic, let's first look at the words *pattern* and *algorithm* separately, before putting them together.

Patterns are present everywhere, certainly in textiles, choreography, mathematics, design and music. However, at first glance, the use of the word *pattern* in music seems comparatively impoverished, at least in the West. At the time of writing, the English language Wikipedia page on pattern has no mention of music, and when musicians talk about pattern, they usually mean any sequence that repeats. The word can even take on a pejorative sense in music, for example in a conference paper on transdisciplinary collaboration Hugill recounts how (and I paraphrase) mathematicians devote their careers to searching for patterns, whereas many composers will be seriously offended if you accuse them of making patterns.[1]

This paper compares patterns in music with those in textiles. The textile arts and crafts are alive with compositional patterning techniques, not only repetitions but reflections and symmetries, and generative structural compositions. Once you scratch the surface of music, it is of course also fully alive with patterns. As just one example, consider *canon* structures, where one voice imitates another, played over the top with a delay - an example of rotational or glide symmetry. So it seems that the difference here is the way words are used; in music, a canon *contains* patterns, whereas in textiles, such a structure would *itself* be referred to as the pattern.

So what *is* a Pattern? Grünbaum and Shephard [12] define patterns as “designs repeating some motif in a more or less systematic manner.” They write in the context of geometric tilings, but the same definition largely holds in music fields; a sequence becomes a pattern, once it is repeated. However, in music we too often focus on the *repetition*, and not the *systematic manner* in which it is repeated. For a pattern to be interesting, we need to do more than repeat it; the repetition only provides the metrical ground on which the pattern acts. Looking around the room you are in now, you will likely see patterns of repetition, but also of reflection, rotation, interference/moire, and glitches or deviations from those patterns. The textiles around (or on) you may well have a visual pattern arising not from the colour of threads alone, but from computational interference between colour and structure.

Accordingly, in the present paper, *pattern* is taken to refer to a whole family of techniques for working with regularities in the world. Such patterns allow us to perceive repetition, reflection and interference in a material. In other words, the way we perceive pattern is inextricably linked with the structured movements of its making. And, once we are dealing with ‘structured movements of making’, we are in the world of algorithms.

An *algorithm* is a step-by-step set of instructions. Sometimes it is assumed that by a step-by-step set, we mean a sequence of instructions, but this is not the case; indeed the notion of an algorithm has been formalised as lambda calculus, which may involve recursive steps *into* a function declaration, rather than stepping *down* a stateful sequence of statements. This clarification becomes important later in this paper, when we address TidalCycles, a live coding environment embedded in the Haskell language, which is itself based on lambda calculus.

There is some sense that as used in this paper the words *algorithm* and *pattern* are synonyms; they both refer to

[1]Hugill's paper is unpublished but a video of its presentation is available at medias.ircam.fr/x6e2d95 with follow-on blog andrewhugill.com/blog/?p=3159 (both accessed 30 Jan 2020).

NIME'20, July 21-25, 2020, Royal Birmingham Conservatoire, Birmingham City University, Birmingham, United Kingdom.

structured ways of making. Therefore the phrase "algorithmic pattern" seems to be a tautology. The phrase is nonetheless useful for on one hand clarifying that we address algorithms not just as software engineering tools, but as formalised ways of making that can to a large extent be perceived in end-results. It also clarifies that we are interested in patterns that are not just simple sequences, but structural qualities. This builds a perspective on pattern as a generative and perceptual connection between creation and reception. In short, I define algorithmic pattern as the perception of systematic activity.

1.1 Patterns in NIME

From her perspective as a foundational algorithmic composer and technologist, Laurie Spiegel [28] argues (in a paper reaching its 40th anniversary) that patterns should be central to computer music interface design, a call relevant to the NIME (New Interfaces for Musical Expression) field. We will return to Spiegel's argument later, but as things stand, how do NIME authors use the word *pattern*? According to my use of the `pdfgrep` utility on a corpus of 1,739 NIME papers[2], the word 'pattern' can be found in 803 papers (46% of the total) of which 315 (18% of the total) contain the word more than twice. I ranked these by incidence, and read the top twenty (which ranged from 26 to 150 occurrences per paper), to gain an impression of how the word is used in the NIME community. Of these twenty papers, I deemed only one [7] explored pattern as an activity, in the context of patterns of interaction emerging and continually changing in motor feedback loops. Of the remaining papers, nine discussed transformation of patterns in some way [5, 13, 14, 16, 19, 24, 31, 33], although all these referred to patterns as being the end result, rather than as a generation/transformation process or notation. The remaining eleven papers [1, 2, 4, 15, 17, 18, 20, 25, 26, 29, 32] referred to patterns as fixed sequences. I identified only one explicit definition of pattern, where Petit et al [25] write "A pattern can be any part of a score, a MIDI sequence, or a pre-recorded sound" – too broad a definition to be useful in the present context.

Wanting to find more examples of papers treating pattern as activity or behaviour rather than sequence, I searched again for the gerund *patterning.* This returned nine papers, none of which had more than two instances of the word. However several went beyond passing references: Green [10] used patterning to refer to the combination of two patterns, Suiter [30] described musical form as a process and patterns in terms of how they relate to one another, Gimenes et al [9] quote Meyer in defining a composer's style as connecting patterns in human behaviour with patterns in results, and my own co-authored paper [3] described live coded algorithms as patterned behaviour. I count all of these examples as fitting the definition of *algorithmic pattern* presented in the present paper, as the perception of systematic activity.

1.2 Algorithmic Pattern in Computer Music Interfaces

Algorithmic and generative music systems often come with high-minded claims of infinite variation or artificial intelligence. However on closer examination, these systems often rely upon surprisingly simple systems based on probability (e.g. Markov chains), arbitrary decisions (randomness/chance) and straightforward sequencing, referred to as 'algorithmic' simply because they are expressed as text rather than as a graphical piano roll. In her essay "Manipulations of Musical Patterns" mentioned earlier, Laurie Spiegel [28] looks beyond such methods, arguing convincingly for greater focus on pattern transformations in computer music interfaces, naming twelve categories of pattern transformation which, she argues, should be as central to computer music interfaces as copy and paste. Many computer music and live coding languages do indeed now feature such pattern languages, including SuperCollider, ixilang, FoxDot, Gibber and TidalCycles.

Our argument is not that algorithmic pattern is complex or difficult, but rather that complexity results from simple parts. In "Notes on Pattern Synthesis", Mark Fell [8] reveals the Max patches behind his acclaimed album *Multistability*, which embrace simplicity in producing intensive pattern studies within self-enforced guidelines. This minimalist approach results in music with clarity, but which is nonetheless complex in structure. The usual minimalist examples, such as Reich's clapping music fit here too, simple in its patterned construction, but bringing forth astonishing variety of detail in its outcome. This is a core benefit to using a pattern as an interface; embracing the simplest ingredients, but transforming them and composing them together to create complex results. Far from new, this approach grounds discussion of music generation in a rich perspective, able to draw from an expansive variety of cultural practices and artefacts from around the world and across history.

When we write something as an algorithmic pattern, we work at least one step removed from the surface of a 'target domain' such as musical notes. By analogy, we don't hit the drum, we write about hitting the drum. *Not even that*, we write about relationships between drum hits, the structures that lead to one movement following another. This is a trade-off, which creates distance between ourselves and an instrument, thereby losing direct tactile control, but which also brings us 'up' to work on a compositional, structural level. Here, we lose physical connection to a drum skin, but instead work in a way where a very small change can create far reaching, often unexpected changes in the music as it unfolds. This generative aspect of algorithmic pattern is what we explore below. Within the limits of this paper, we are unfortunately unable to explore this expansive world in-depth, which encompasses the history of all arts and crafts. Instead we will focus on two examples: first, live coded patterns in music, and then woven patterns in textiles.

2. ALGORITHMIC PATTERN IN TIDALCYCLES

Work on TidalCycles (commonly *Tidal* for short) first began around 2009, and over the past decade has developed into a comprehensive, free/open source environment for algorithmic pattern, mainly in the context of live coding music. At heart, it is a domain specific language (DSL) and environment for patterning Open Sound Control network messages, embedded in the pure functional language Haskell. Tidal is usually used in tandem with SuperDirt, a hybrid framework for sample manipulation, synthesis and MIDI, implemented in SuperCollider. However, Tidal can be applied to any kind of pattern, and has indeed been used to pattern live choreographic scores [27], woven textiles [23], DMX-controlled lighting, and VJing. Indeed, in sympathy with the present medium, all the below examples in this section use Tidal to create visual rather than musical pattern.

While Tidal has been developed alongside creative practice, it upholds strong computer scientific principles. Cru-

[2] Pdfgrep is available at pdfgrep.org. It was used with `-i` and `-c` parameters, for case-insensitive matching. For example, the top twenty in terms of number of matches was found with the following: `pdfgrep -i -c pattern *pdf | sort -n -t: -k2 | tail -n20`

cially, a pattern is defined as a pure function, and therefore may be composed with other patterns flexibly and safely. As Tidal has developed, its core representation has grown more succinct, and a recent rewrite resulted in more rigorous understanding of what, as far as Tidal is concerned, a pattern *is*.

2.1 Tidal type structure

Tidal's notion of pattern follows from its representation within Haskell's type system, as a pure function of time. This follows work on pure functional reactive programming [6], where rather than representing data using lists, behaviour is represented with functions. Accordingly, rather than representing a sequence as a list of events, Tidal represents it as a function taking a timespan as input, and then returning all the events that are active during that timespan. In this way, the idea of a pattern being about behaviour rather than sequence is embedded in Tidal's core.

Let's have a look at the type declarations themselves, describing each for those unfamiliar with the Haskell language.

```
data Arc = Arc {start :: Rational, stop :: Rational}
```

A timespan is expressed as an *arc* of time, consisting of a start and end time. Timespans are referred to as arcs, in sympathy with Tidal's cyclic notion of time. Importantly, time is represented by a rational number, thereby allowing time to be arbitrarily subdivided, without any loss of accuracy (that the more common representations based on floating point numbers are known for).

```
data Event a = Event {
    whole :: Maybe Arc,
    part :: Arc,
    value :: a
}
```

An *event* contains a value of some type a, and two arcs. The *part* arc represents the timespan during which the event is active. An event may represent part of a larger *whole* timespan, which is the second arc. If a whole is not set, this indicates that the event is *continuous*; that is, rather than having a discrete beginning and end, the value is able to change continuously. When querying a continuous value, the result is sampled from the midpoint of the query arc. This approach allows both discrete and continuously varying events to co-exist in the same pattern.

```
type Query a = (Arc -> [Event a])
data Pattern a = Pattern {query :: Query a,
                          controls :: StateMap
}
```

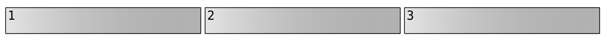
A *query* represents the pattern's behaviour, as a function from time arcs to events. In particular, a query takes an arc as input, and returns a set of events which are active during that arc as output. Event values in a given pattern must all be of the same type, and the 'part' arcs of the events will be constrained to the query arc. If an event 'whole' extends beyond the query, it is returned as-is, but its 'part' is curtailed. In other words, when a query returns a fragment of an event, the caller is also given the 'whole' arc of which the fragment is part.

2.2 Tidal composition

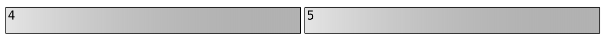
The above definition of pattern does not say much about Tidal as an interface, but what follows from it is a rich approach to composition, supported by a large library of pattern combinators. *Composition* is meant here in both a musical and computer scientific sense, in terms of composing together musical behaviours into new, generally more complex behaviours. Tidal supports a multitude of ways to combine patterns together, many based on Tidal's allowing patterns to be treated as values; that is, any function that combines two values, can be used to combine two *patterns* of values.[3]

As a trivial example, let's combine two tidal patterns `fastcat [1,2,3]` and `fastcat [4,5]`. The first thing to note is that `fastcat` combines a list of patterns into a contiguous sequence, of equal durations over a cycle. The metrical cycle is in general the reference point in a Tidal operation, rather than a beat or step. Therefore, we need to combine two patterns with different structures - one has three events per cycle, and the other has two. We can visualise them like this:

```
fastcat [1,2,3]
```

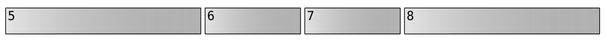


```
fastcat [4,5]
```



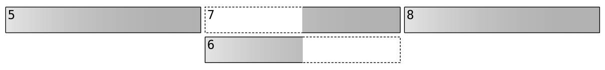
Patterns continue into infinity, but in the examples here we visualise just the first metrical cycle. We can combine these two patterns by adding them together with +:

```
fastcat [1,2,3] + fastcat [4,5]
```



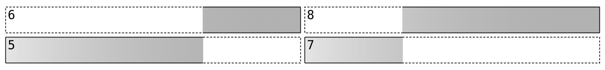
In the above, 1 gets added to part of 4, 2 gets split between 4 and 5, and 3 gets added to part of 5. An alternative operator, |+, privileges structure on the left. The same events are matched up, but the resulting events maintain the 'wholes' from the pattern on the left hand side of the operator:

```
fastcat [1,2,3] |+ fastcat [4,5]
```



Event fragments are shown with their active parts shaded within their original 'whole'. Conversely, the +| operator privileges structure from the right hand pattern:

```
fastcat [1,2,3] +| fastcat [4,5]
```



Note that when such a pattern structure reaches the scheduler, only the events that have their onsets intact will result in an event actually being triggered. That is, the start of an event's 'part' must be the same as that of its 'whole' to result in sound, otherwise it represents a fragment of an event's tail, only useful for combining with other events.[4] Accordingly, the first example would trigger four sounds, the second three sounds, and the final one two sounds.

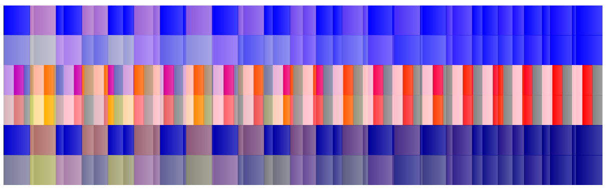
Tidal has a set of such operators for basic arithmetic, but any function can be used to combine patterns together in this way by using Haskell's standard syntax for applicative

[3]In other words, in functional programming terms, in Tidal a pattern is an applicative functor.

[4]A model that works beyond trigger messages, allowing continual varying of a sound's parameters after it has been triggered, is at working prototype stage.

functors, with Tidal's additional nonstandard operators `<*` and `*>` for privileging structure on the left or right. For example, to merge two colour patterns using the `blend` function, with an additional continuous sine pattern to control the blend from one pattern to another:

```
blend
  <$> (slow 4 sine)
  *> "[blue, pink red grey, darkblue]*20"
  <*> "{blue orange, darkgrey yellow pink}*11.5"
```



Instead of using 'fastcat', the above makes straightforward use of Tidal's mini-notation for polyrhythmic sequences, denoted by double quotes.

It is worth reiterating at this point that all these patterns are functions, and not data structures. By combining them in this way we are not computing anything, only creating a new function composed of other functions, i.e. composing behaviours. No calculation actually takes place until the resulting pattern is queried.

2.3 Patterned parameters

TidalCycles has a large library of combinators, but for the purpose of this paper we will focus on just one, the `fast` function, which simply speeds up (or for factors < 1 slows down) a pattern. Its definition is minimal, taking a time factor and a target pattern as input, and manipulating the target pattern's timeline according to the factor:

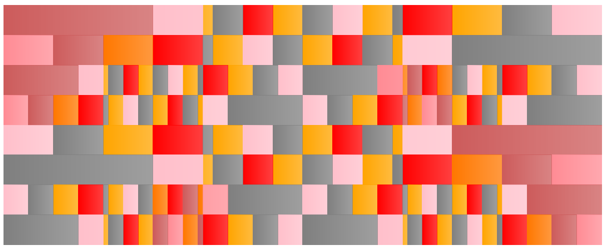
```
fast timepat pat =
  innerJoin ((\time -> withResultTime (/ time) $
                       withQueryTime (* time) pat)
             <$> timepat
           )
```

What is interesting in the above is that the time factor input is itself a pattern. With combined use of the `<$>` operator and `innerJoin` function, we manipulate the target pattern *inside* the pattern of time. This higher order magic uses much the same procedure to combine patterns as the one described earlier. The result is a highly flexible function for patterning the speed of a pattern. For example:

```
fast 4 $ fast "1 2 3" "grey pink red orange"
```

The above switches between slices of the colour pattern running at different speeds. An additional `fast` function is applied so that you can see four cycles of the result. Once a few more simple transformations are added, textures begin to form:

```
superimpose rev $ superimpose (fast 2)
  $ chunk 4 (blend 0.5 red <$>)
  $ superimpose rev
  $ fast "1 5 3"
  $ iter 4 "grey pink red orange"
```



Again, the above code does not calculate anything on its own, it composes together into a single function, which is then passed to the scheduler (or in this case, graphics renderer) which queries the required time arcs.

2.4 TidalCycles as algorithmic pattern interface

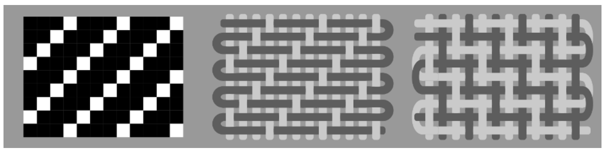
Please refer to tidalcycles.org for further details of Tidal's library of combinators, polyrhythmic mini-notation, and independently patternable effects. But already, we can see some of the affordances which Tidalcycles offers. Every part of the above code example is trivial on its own, starting with a four-step sequence, and adding simple transformations on top. However, the results quickly become astonishingly complex, with each edit giving results which become practically impossible to predict. This is because the different elements interfere with each other, so that every simple part has complex influence over the whole.

3. ALGORITHMIC PATTERN IN HAND WEAVING

Hand weaving is an advanced world of technology, having developed over thousands of years within diverse cultures of practice across the world. All weaving is digital technology, in that it involves discrete crossing points between warp (vertical) and weft (horizontal) threads. As with Tidal, a weave involves interference between multiple interacting systems. In other words, all weavers work with algorithmic patterns, *especially* hand weavers, who work without any machine or computer to do calculations for them, so do it all themselves.

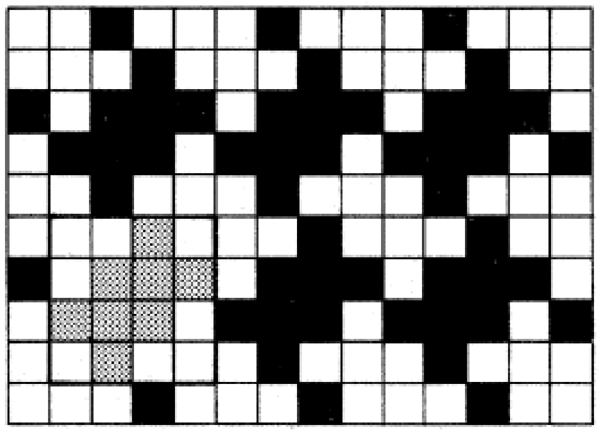
3.1 Colour and weave

It is difficult to get across the complexities of weave in a single paper, but one clear example is the family of colour-and-weave effects, where systems of colour and binary structure interfere to create the end result [22]. The following shows an elementary example with a diagonal 'twill' structure. The left shows the weave 'block' structure, a binary grid representing meeting points between warp and weft threads. A square is black where a weft goes over the warp, and white where it goes under. The central diagram shows a simulation of this pattern woven with light weft and dark warp threads, with the diagonal structure visible in the results. However, if we weave the same structure with alternating light and dark threads on both the warp and weft, the result (shown on the right) has the appearance of a diagonal moving in a different direction. This is a trivial example of a colour-and-weave effect, which can be exploited to create a wide range of imagery from the same grid structure.

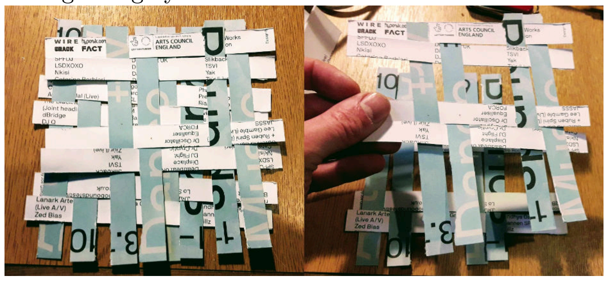


3.2 Double weave

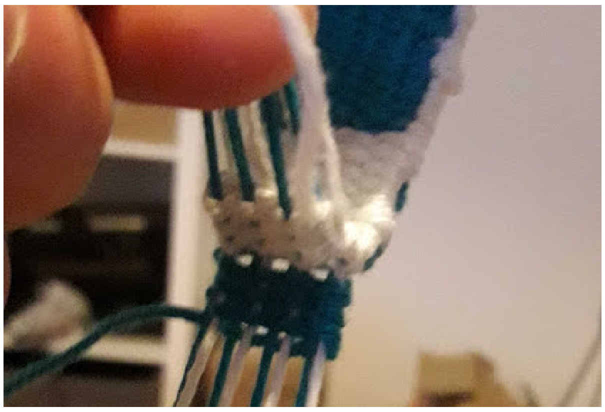
Working with weaves as a two dimensional binary structure is a useful abstraction, but like all abstractions is not a complete model. The threads of weaves move in three dimensions, with the structure of the threads themselves, and their behaviour under tension, having strong bearing on the end result. It is therefore a mistake to think that the binary grid fully represents reality. Such a mistake seems to have been made by Grünbaum and Shephard [11], who approach weaving from the point of view of two-dimensional tiling patterns. Using geometric rules, they prove that the following weave structure result in a textile which 'falls apart':



When this structure is 'woven' using rigid card, this does appear to be the case; half of the weave lifts off the rest, resulting in highly unstable structures:



However as can be seen below, if we weave the structure with warp threads under tension on a loom, we find that the two layers hold together perfectly. Rather than 'falling apart', the textile simply splits in two. In fact, this technique of weaving two (or more) layers at once is very well known by weavers as double (or triple, etc) weave, resulting in a thick structure, where threads from the two layers can exchange to create a range of effects.[5]



[5] The weave was itself live coded, on the 'live loom' [21]

4. ALGORITHMIC PATTERN AS INTERFACE

To conclude, let's consider the place of algorithmic pattern as an interface between a musician and their music. We have seen how the algorithmic patterning of interference patterns within a two-dimensional grid acts as an interface between weaver and weave. It allows manipulation of textile at one-step removed, in terms of higher order structure that can generate surprising results, including through colour-and-weave and double weave effects. But when weaving we must recognise that the block design is only one part of the whole, and that to weave is both a computational and embodied experience where abstract algorithmic patterns meet real-world behaviours. It is not possible to understand a woven structure without actually weaving it.

The same lesson applies to algorithmic patterns in live coding. We can work with the pattern as code, but it does not notate what we hear and feel. Not only do interference patterns work inside the computer at scales beyond our imaginations, but they then leave the computer as sound, perceived as music in ways which do not exist in the notation but in our embodied minds. Live coding involves an improvised movement of pattern across cognition, computation and perception, a fundamentally experimental activity, where code is developed in the open-minded and open-bodied spirit of discovery. Without understanding that the algorithm is only one step in the creation of music, we might find that our music simply falls apart.

5. ACKNOWLEDGMENTS

This research is conducted by the PENELOPE project, with funding from the European Research Council (ERC) under the Horizon 2020 research and innovation programme of the European Union, grant agreement No 682711.

6. BIBLIOGRAPHY

[1] Bois, A.R.D. and Ribeiro, R.G. 2019. HMusic: A domain specific language for music programming and live coding. *Proceedings of the international conference on new interfaces for musical expression* (Porto Alegre, Brazil, 2019), 381–386.

[2] Bouillot, N. 2007. NJam user experiments : Enabling remote musical interaction from milliseconds to seconds. *Proceedings of the international conference on new interfaces for musical expression* (New York City, NY, United States, 2007), 142–147.

[3] Collins, N. and McLean, A. 2014. Algorave: Live performance of algorithmic electronic dance music. *Proceedings of the international conference on new interfaces for musical expression* (London, United Kingdom, 2014), 355–358.

[4] Derbinsky, N. and Essl, G. 2012. Exploring reinforcement learning for mobile percussive collaboration. *Proceedings of the international conference on new interfaces for musical expression* (Ann Arbor, Michigan, 2012).

[5] Eigenfeldt, A. and Kapur, A. 2008. An agent-based system for robotic musical performance. *Proceedings of the international conference on new interfaces for musical expression* (Genoa, Italy, 2008), 144–149.

[6] Elliott, C. 2009. Push-pull functional reactive programming. *Proceedings of 2nd acm sigplan symposium on haskell 2009* (2009).

[7] Faubel, C. 2014. Rhythm apparatus on overhead. *Proceedings of the international conference on new interfaces for musical expression* (London, United Kingdom, 2014), 491–494.

[8] Fell, M. 2018. Notes on pattern synthesis: 1983 to 2013. *Oxford handbook on algorithmic music*. A. McLean and R. Dean, eds. Oxford University Press.

[9] Gimenes, M. et al. 2007. Musicianship for robots with style. *Proceedings of the international conference on new interfaces for musical expression* (New York City, NY, United States, 2007), 197–202.

[10] Green, O. 2014. NIME, musicality and practice-led methods. *Proceedings of the international conference on new interfaces*

for musical expression (London, United Kingdom, 2014), 1–6.

[11] Grünbaum, B. and Shephard, G.C. 1980. Satins and twills: An introduction to the geometry of fabrics. *Mathematics Magazine*. 53, 3 (1980), 139–161.

[12] Grünbaum, B. and Shephard, G.C. 1986. *Tilings and patterns*. W. H. Freeman & Co.

[13] Hawryshkewich, A. et al. 2010. Beatback : A real-time interactive percussion system for rhythmic practise and exploration. *Proceedings of the international conference on new interfaces for musical expression* (Sydney, Australia, 2010), 100–105.

[14] Jordà, S. et al. 2016. Drumming with style: From user needs to a working prototype. *Proceedings of the international conference on new interfaces for musical expression* (Brisbane, Australia, 2016), 365–370.

[15] Kim, T. and Weinzierl, S. 2013. Modelling gestures in music performance with statistical latent-state models. *Proceedings of the international conference on new interfaces for musical expression* (Daejeon, Republic of Korea, May 2013), 427–430.

[16] Kitani, K.M. and Koike, H. 2010. ImprovGenerator : Online grammatical induction for on-the-fly improvisation accompaniment. *Proceedings of the international conference on new interfaces for musical expression* (Sydney, Australia, 2010), 469–472.

[17] Lee, E. et al. 2007. Towards rhythmic analysis of human motion using acceleration-onset times. *Proceedings of the international conference on new interfaces for musical expression* (New York City, NY, United States, 2007), 136–141.

[18] Lee, E. et al. 2006. Conga: A framework for adaptive conducting gesture analysis. *Proceedings of the international conference on new interfaces for musical expression* (Paris, France, 2006), 260–265.

[19] Lee, S.W. et al. 2012. Real-time music notation, collaborative improvisation, and laptop ensembles. *Proceedings of the international conference on new interfaces for musical expression* (Ann Arbor, Michigan, 2012).

[20] Lui, S. 2014. A real time common chord progression guide on the smartphone for jamming pop song on the music keyboard. *Proceedings of the international conference on new interfaces for musical expression* (London, United Kingdom, 2014), 98–101.

[21] McLean, A. 2020. The Live Loom. *Proceedings of the 5th International Conference on Live Coding* (Limerick, Ireland, 2020).

[22] McLean, A. and Harlizius-Klueck, E. 2018. Digital art: A long history. *Proceedings of the international conference on live interfaces* (2018), 71–77.

[23] McLean, A. and Harlizius-Klück, E. 2018. Fabricating algorithmic art. Austrian Cultural Forum.

[24] Ogawa, K. and Kuhara, Y. 2009. Life game orchestra as an interactive music composition system translating cellular patterns of automata into musical scales. *Proceedings of the international conference on new interfaces for musical expression* (Pittsburgh, PA, United States, 2009), 50–51.

[25] Petit, B. and serrano 2019. Composing and executing interactive music using the hiphop.js language. *Proceedings of the international conference on new interfaces for musical expression* (Porto Alegre, Brazil, 2019), 71–76.

[26] Schoonderwaldt, E. and Jensenius, A.R. 2011. Effective and expressive movements in a french-canadian fiddler's performance. *Proceedings of the international conference on new interfaces for musical expression* (Oslo, Norway, 2011), 256–259.

[27] Sicchio, K. 2014. Data management part III: An artistic framework for understanding technology without technology. *Media-N: Journal of the New Media Caucus*. 10, 3 (2014).

[28] Spiegel, L. 1981. Manipulations of musical patterns. *Proceedings of the symposium on small computers and the arts* (1981), 19–22.

[29] Subramanian, S. et al. 2012. LOLbot: Machine musicianship in laptop ensembles. *Proceedings of the international conference on new interfaces for musical expression* (Ann Arbor, Michigan, 2012).

[30] Suiter, W. 2010. Toward algorithmic composition of expression in music using fuzzy logic. *Proceedings of the international conference on new interfaces for musical expression* (Sydney, Australia, 2010), 319–322.

[31] Toka, M. et al. 2018. Siren: Interface for pattern languages. *Proceedings of the international conference on new interfaces for musical expression* (Blacksburg, Virginia, USA, 2018), 53–58.

[32] Trail, S. et al. 2012. Non-invasive sensing and gesture control for pitched percussion hyper-instruments using the kinect. *Proceedings of the international conference on new interfaces for musical expression* (Ann Arbor, Michigan, 2012).

[33] Vogl, R. and Knees, P. 2017. An intelligent drum machine for electronic dance music production and performance. *Proceedings of the international conference on new interfaces for musical expression* (Copenhagen, Denmark, 2017), 251–256.