Algorithms and computation in music education

Andrew R. Brown
Griffith University

Abstract

The chapter will discuss how bringing music and computation together in the curriculum offers socially grounded contexts for the learning of digital expression and creativity. It will explore how algorithms codify cultural knowledge, how programming can assist students in understanding and manipulating cultural norms, and how these can play a part in developing a students' musicianship. In order to highlight how computational thinking extends music education and builds on interdisciplinary links, the chapter will canvass the challenges, and solutions, involved in learning through algorithmic music. Practical examples from informal and school-based educational contexts are included to illustrate how algorithmic music has been successfully integrated with established and emerging pedagogical approaches.

Keywords

Music, algorithm, education, computer, learning

Introduction

Music is often described as "organised sound", and improving students' understanding about that organisation, as part of developing their musicianship, is a typical educational goal. An algorithmic description of musical processes can contribute to such development by requiring students to externalise and formalise their understanding. Programming and playing back the results of those algorithms provide rapid feedback, allowing reflection on and refinement of ideas. Further, the design of musical algorithms serves both to demonstrate understanding and to provide a conduit for creativity.

This chapter will explore ways in which algorithms and coding skills can be useful intellectual tools that assist in the development of musical intelligence and computational thinking. There are clear lessons for computer science educators here too; however, the emphasis in this chapter is on music education, given that elsewhere the role of music, and audio-visual media more generally, in enhancing the study of computer programming is quite well covered. The interested reader is directed to references such as *Mindstorms: Children, Computers, and Powerful Ideas* (Papert 1980); *Turtles, Termites, and Traffic Jams* (Resnick 1994); *Changing Minds: Computers, Learning, and Literacy* (diSessa 2000), *Introduction to Computing and Programming in Python: A Multimedia Approach* (Guzdial and Ericson 2010) and *Making Music with Computers: Creative Programming in Python* (Manaris and Brown 2014).

Music education is concerned with all aspects of music, including: listening, composition, performance, analysis, critique, recording, distribution, and cultural awareness. Algorithms and their computational implementation have the potential to be applied to many of these areas of study, from models of music perception, through analytical techniques of empirical musicology and music information retrieval, to computer assisted composition and interactive performance systems.

In his article, 'Computation and Music', Heinrich Taube (2012) distinguishes between three levels of computational representations of music. Firstly, the *acoustic* level is identified where sound waves, synthesis and physical properties of sound, space and instrumentation are described and manipulated. Secondly, the *performance* level involves score interpretation and

physical sound control gestures. At this (and the third) level, stylistic, cultural and equipmental conventions and constraints must be accounted for. Thirdly, the *compositional* level is concerned with structural organisation and musical elements and events. At this level music is often represented as symbolic notation and structural relations. Taube emphasises the significance of abstraction between these levels and the general role of abstraction in digital representations of music that are subject to computational algorithms. Describing abstractions as ‘metalevels’ he writes, ‘the metalevel representation[s] are active representations of the processes, methods, algorithms, and techniques that a composer develops to craft the sounds of his or her compositions’ (Taube 2004, 3).

Abstractions, and their intellectual potential, have been clearly articulated as key to computer science education at least since Papert’s (1980) *Mindstorms: Children, Computers, and Powerful Ideas*. In their landmark programming text, Abelson and Sussman begin by emphasising that programming is about ‘building abstractions and procedures’ (1996, 1).

The understanding of musical patterns as abstractions of cultural practices (e.g., musical form) or as characteristics of cross-cultural sonic expression (e.g., auditory streaming) has been a mainstay of music education—although patterns (e.g., arpeggios) or processes (e.g., serialism) are not typically framed as algorithms, but more commonly as theories, models or techniques. Nevertheless, the connections are not difficult to see. Composers were using algorithms well before the advent of computers to assist in exploring musical space to generate novel material and ideas (Nierhaus 2010) and with the advent of computing this has only expanded; from Xenakis to Eno. There is much to be gained by training musicians in the applications of algorithmic process, not only to help them understand existing musical processes, but also to prepare them to push the boundaries of musical possibility and help transform musical experiences. These are important objectives for music education (Jorgensen 2003; Dillon 2007).

Just like music educational practices in general, the application of algorithmic processes are often focused on music creation and presentation. This will become evident in the examples presented later in the chapter. However, algorithmic representation and design emphasises a particular analytical engagement with music. As the music educator Keith Swanwick points out; ‘drawing attention to certain features of music is inevitably a form of analysis... It is simply a way of picking out patterns from an overall impression, for instance by focussing on such things as melodic development, harmony or instrumentation’ (1994, 12-13). The description of music in algorithmic terms is an effective way of focussing attention on the organisation of music.

This chapter will explore the implications of an algorithmic approach to music learning. It will then discuss a number of the challenges and opportunities that such an approach presents to existing educational practices. Finally, a series of examples reveals the variety of approaches that innovative educators have employed to introduce algorithmic music as part of their programs.

Automation and Agency

Ever since Pythagoras there have been links between music and mathematics. As a result, the systematic description of processes that lead to sonic and musical structures is deeply embedded in our culture. The algorithmic description of these processes has long been applied to technological music making in the form of instrument design and to various mechanical music devices such as the music box and player piano (Levenson 1994; Collins this volume). Computational descriptions of algorithmic processes are just the most recent, and also most powerful, application of this link between music and mathematics.

Programmability makes computers outstanding automators of musical processes and the autonomy that results provides computers with an unprecedented degree of agency in the music making that ensues. In these ways, algorithmic musical processes provide rich opportunities for enhancing music learning through automated support, through the articulation and reflection of

musical structures, and via the codification of innate capabilities and cultural musical conventions and behaviours.

Scaffolding and Access

Generative music systems rely on algorithmic descriptions for the real-time creation of music. The resulting outcomes often include variations at each generation, depending upon the amount of indeterminism in the processes. This provides an interesting parallel to the variability of interpretations that human performers provide, which is differentiated from fixed recordings were the dominant mode of music delivery in the past 100 years (see Levton this volume). Educating students about these new generative and interactive music methods is increasingly important. But generative systems can also provide support for developing traditional musical skills.

Because an algorithmic system can generate music autonomously it can be a useful scaffold for beginner musicians; they can either play along with it, or 'direct' it through parametric control. As a result, students can be part of a musical outcome much richer than they alone might be able to produce—with obvious benefits to levels of motivation and self-esteem. The use of technologies in scaffolded learning has been well documented as a useful pedagogical strategy (Luckin 2008). Systems such as the authors' Jam2jam tool (see Figure 1) have allowed novice musicians to engage with musical concepts well beyond what would be accessible with their limited acoustic instrumental skills.

Figure 1. The Jam2jam AV interactive audio-visual system employs generative music algorithms.

Interactive music systems, such as the numerous apps for mobile platforms, or the various music games, provide ready access to musical interactions because of their low skill requirements. From a learning perspective this can provide a shallow 'on ramp' to engagement with music and, it is hoped, spark an ongoing interest in music and the developing of further musical skills. The same accessibility also makes algorithmic music systems ideal for those with special needs (Adkins et al. 2012).

Description and Notation

Beyond the *use* of algorithmic music systems by students is their involvement in system *design and development*. It has long been recognised that more can be learned by teaching a topic than by studying it. So, 'teaching' a computer to make music, by programming it to follow an algorithm, has a similar benefit.

Typically, algorithmic design and deployment require one to describe the process and then to codify it in a form the computer will understand. The initial design is often in a human readable form such as a diagram, a description written in prose, or a list of steps similar to a recipe. Prototypes can be created for manual testing, or steps might be performed in a digital audio workstation to explore procedures before coding. Programming is the task of articulating the algorithm in code; it requires the description to be rendered in a notation for the computer to interpret—usually a programming language. Like other musical notations (typically staff notation) the code description can be considered as a musical score; in the case of code, a score for the computer to follow.

Following Taube's three levels of computational representations of music, algorithms can be created to manage musical arrangements, compositions, performance renderings, or sound design; these tasks fit neatly into existing music education curricula. Coding can be considered an alternative notation just as graphic scores are. There are many compositional precedents in the use of alternative notations, including the instruction-based scores used by John Cage and other composers in the twentieth century. Algorithmic musical processes can be contextualised as part of the evolution of musical processes and have a logical place in a holistic view of notational literacy, as they are used in music texts like David Cope's *New Directions in Music* (2001).

Biological and Cultural connections

Interacting with and designing algorithmic music systems involves immersion in sonic and musical conventions—even if it is to countervail them. These conventions are often cultural and they vary, in either dramatic or subtle ways, between cultures and within subcultures. At times, constraints are imposed on music making by our biological condition; for example, we have two ears and particular perceptual capabilities, and we embody certain motor skills and capabilities. At times our music algorithms need to mimic these constraints to fit in with musical conventions, or they might extend the musical opportunities available beyond those boundaries.

Learning music through engagement with algorithmic processes allows students to undertake activities such as simulating the cyclic patterns of Indonesian Gamelan (see Matthews this volume), exploring the limits of drum kit performance using four 'limbs', and using subtle variations in timbre, space and temporal alignment to investigate the perceptual boundaries of polyphony and musical textures. Along the way, students learn to understand the rituals, tuning systems and other aspects of these musical practices. For example, Gamelan performance, or how typical drumming patterns are modulated by part coordination, and how psychoacoustic phenomena such as audio streaming and spectral masking affect the reception of music.

Challenges and Solutions

Given the many opportunities for algorithmic music to play a positive role in music education, as outlined in the previous section, it might seem odd that it is not already a standard part of all music education courses. Clearly, there are challenges involved in the adoption of algorithmic music activities for learning. These include the fact that often teachers are not experienced in algorithmic music, and that sometimes algorithmic methods are just a tedious route to mediocre musical outcomes, and therefore uncompetitive with traditional approaches. Examples and activities must be chosen with care to ensure that the relevance and the value of algorithmic approaches are maintained.

Another disincentive is that established disciplinary boundaries, often reinforced by educational institutions, mean that coding is perceived as a computing skill rather than a musical one, and therefore outside the responsibility (or even legitimacy) of a music education. As we will see in many of the examples described later in this chapter, a common approach to overcoming this division of responsibility is interdisciplinary collaboration, where music becomes a motivating context for learning programming and/or programming becomes an avenue to enhanced music making.

Fear of the unknown

For this author, and probably most authors in this volume, algorithmic music brings together multiple passions across the creative arts, technology and more. That same interdisciplinary proclivity, however, might serve as a barrier to entry for many others. Often musicians love music because it is an escape from maths and science; conversely, many computer scientists and engineers might feel uncomfortable with the ambiguities inherent in the creative arts, let alone the immateriality of music as an art form. Fear of the unknown and lack of self-confidence outside an established domain of understanding can be a significant barrier to educators' and students' serious engagement with algorithmic music.

Apprehension about the unfamiliar is not a new phenomenon in education and many approaches have been tried to overcome these barriers. Amongst them is the use of very structured tasks that lead people step-by-step through what might be unfamiliar territory. Brief excursions (small tasks) can also provide stepping-stones to more in-depth engagement—for example, the use of exercises that require only short fragments of code, or tasks that do not rely on an advanced music theory background. Group projects allow people to share the journey and provide peer support when the inevitable challenges arise; they also allow for ensemble experiences, which are an important aspect of musical training. Mentoring and exposure to stories from experienced travellers help to show that the journey to understanding and using algorithmic music is possible and worthwhile. Exemplars can demystify the creative and/or technical process as one requiring only persistent and iterative steps rather than involving a mystical leap of understanding.

Educators, and certainly their students, are well aware that they live in a technological society and that the role of technology in music is significant. The music industry, after all, has been at the leading edge of disruption from digital technologies, both in the way music is made and how it is consumed. What seems less well appreciated, however, is the role of programming in driving that technological society and the opportunities for musicians who can code to increase control over their technological destiny. Two educational movements that have tried to help people learn to work with algorithms are courses in Computational Thinking and in Creative Coding.

Computational Thinking and Music

Courses in Computational Thinking introduce students to methods of description and design that follow algorithmic processes without immediately engaging in the representation of these as programming notations. Rather, algorithmic solutions to tasks are approached through familiar and tangible media such as writing, drawing, block building, and so on. When code is introduced, tasks often maintain a media orientation through computer graphics, sensor-based electronics, robotics, and sound. This approach to understand computing before attempting to code is designed for a non-specialist audience (such as musicians), and is deliberately different from what might be considered appropriate for computing professionals (Guzdial 2008).

The main conceptual building blocks explored in computational thinking courses are abstraction and automaton (Wing 2008). In music, abstractions include structural patterns such as canons, counterpoint, phasing and so on, and the automation of these can be manual, mechanical, or computational. As Edwards notes: 'That algorithmic thinking is present in Western composition for at least 1,000 years has been established. That such thinking should lend itself to formalization in software algorithms was inevitable' (2011, 67). Similarly, algorithmic thinking about music is a logical development of existing theoretical and analytical approaches to music and, despite some reticence about the value of computational thinking curricula (Denning 2009), it seems reasonable to expect that approaches used for the teaching of computational thinking might be applied to thinking algorithmically about music (Ruthmann et al. 2010).

Creative Coding Movements

The application of computational processes in the creative arts has gathered momentum since at least the 1990s when New Media Art or Multimedia courses championed the production of digital creative works; most often these were screen based works or animated installations. In more recent years, with the popularity of dedicated programming environments such as Processing (Reas and Fry 2003) and the increasing ubiquity of JavaScript automation in web sites, artistic computational works have become popular under the banner of *creative coding*.

A leading early figure in this movement was John Maeda who led the Aesthetics and Computation Group at the MIT Media Lab around the turn of the century. In his book, *Creative Code*, he outlines the shift in emphasis for creative artists, in which computer technology 'is not a tool; it is a new material for expression' (2004, 101). It is this shift in emphasis from using digital technology as a tool to understanding digital media as a medium that underscores creative coding activities—a shift for creative artists from technology *use* to technology *craftsmanship*.

A direct outcome of Maeda's ideas was the Processing environment, created by two of his students, which sparked an expansion of creative coding courses in art and design colleges around the world. It includes built in, and third party, music and audio libraries that allow for algorithmic music activities such as building interactive sound scapes, or creating a step sequencer. Other accessible creative coding environments with a strong music focus include Fluxus (Griffiths 2007) (<http://www.pawfal.org/fluxus/>), Gibber (Roberts 2013) and those used in the examples discussed below.

Learning Contexts

Appropriate settings for learning about algorithmic music are many and varied, covering formal and informal learning contexts (Green 2002). Informal settings include community run 'hacker' sessions, 'maker' workshops, and after-school computer sessions, including the Computer Clubhouse (Sylvan 2006) and Code Club (Code Club World 2015). Formal settings mainly

include courses at schools, colleges, and universities, but increasingly include offshoots of these available freely as online courses (e.g., Kapur 2013; Freeman 2014; Dave Conservatoire 2016).

Algorithmic processes can be applied to many areas of music making, as can be seen by the variety of perspectives in this volume. Any of these areas of activity can provide a context in which students can hear about algorithmic music. The most well established context is the use of algorithmic processes in composition, where they are used to either generate material and/or where algorithmic processes are integrated as part of the realisation of the work itself. Related to this, algorithmic music and/or sound design can be used in art installations and interactive media such as computer games.

Interactive music devices (instruments) that incorporate algorithmic processes have become quite popular over the past decade. Many of these are featured at the successful New Interfaces for Music Expression (NIME) conference series (<http://nime.org>). As mobile computing power has increased, algorithmic music has become a part of live performance practices. A prominent example is the live coding community that is rapidly expanding (see Roberts this volume). Live coding practice includes solo and ensemble performances with music and/or audio-visual outcomes. A number of educational programs have included live coding in their Laptop Orchestras, where students make music with code as an ensemble (see Ogborn this volume). A wide range of musical genres is represented in live coding practices, including experimental, electroacoustic, electronic dance music, and neo-classical.

Real-time programming environments are necessary for live coding and are useful for learning about algorithmic music. Real-time programming environments allow code to be updated while it is running, enabling changes to be made on the fly with an uninterrupted flow of musical output. This immediacy of audible feedback is typical of interaction with acoustic instruments and sound in the physical world, but has until recently been uncommon in computer programming workflows, where applications are typically halted and restarted after editing.

Examples

In this section, there are descriptions of several examples in which algorithmic music has been used in educational contexts. These exemplars will show how the issues and concerns discussed so far are managed in the context of real-world learning situations.

TuneBlocks

Jeanne Bamberger is Professor Emeritus of Music and Urban Education at the Massachusetts Institute of Technology. She designed the TuneBlocks compositional environment as a result of research that combined music psychology and computers in education. The software allows for the creation of musical fragments (blocks) and their combination in series and hierarchies. TuneBlocks was designed to support analysis (through reconstruction of existing works) and creation (through elaboration/creation of new material). First developed in the 1980s as part of Bamberger's Impromptu software, a version of TuneBlocks is available today for the Apple iPad (Figure 2). While its algorithmic capabilities are quite limited by today's standards, it is an important landmark in the use of computers and algorithmic thinking in music education.

Figure 2. TuneBlocks' playroom interface on the Apple iPad.

Individual tune blocks are visualised as square icons each representing a musical motif (see figure 2). This abstract representation is designed 'to focus the students' attention on *listening* rather than looking' (Bamberger 2003, 11). A common listening task with TuneBlocks is to have a melody divided amongst a set of blocks that a student must rearrange in the correct order. Extension activities can include arranging blocks in a new but effective order, and composing additional blocks to extend further the rearrangement and compositional possibilities. Through these recombinatorial processes, Bamberger's objective is to have students learn, through listening and experimentation, how to discern specific features of each block, how some are similar and others differentiated, what are the structural functions of each block (beginning, middle, end, and so on), why blocks combine well or not, and to appreciate how order and repetition matter. Students can also reflect on why they like or dislike particular blocks or combinations of blocks (Bamberger 2003).

While automation in the TuneBlocks application is limited, it is designed to support what is now called computational thinking. The simple interface was designed for children to use in an age when computing was not nearly as ubiquitous as it is today, and grew out of earlier experiences Bamberger had with musical coding in the Logo language, in the 1970s, when she collaborated with Seymour Papert and other early pioneers of teaching computer programming in schools (Bamberger 1979).

The notion, clearly evident in Bamberger's work, of developing understanding through practical activity is sympathetic with ideas of constructivist psychology. According to this theory, people develop and transform their understanding and ideas through experiences in the world, and in doing so they construct and internalise new knowledge. Constructivism is based on the developmental psychology theories that Jean Piaget established during the mid-1900s.

Bamberger's colleague, Seymour Papert, was a student of Piaget. It was natural that these researchers would see the potential in computer programming and interface design for the articulation and externalisation of processes and structures. Computers and computation, therefore, were taken up as vehicles for developing the kinds of systematic (algorithmic) thinking that are evident in many fields, including music composition.

PowerBooks_UnPlugged

Laptop computer performance ensembles are increasingly common in educational settings, particularly at university level. One example is PowerBooks_UnPlugged, based at the Institute for Time Based Media at the University for the Arts in Berlin. The ensemble's web site declares: 'Many have claimed that "The laptop is the new folk guitar"; if this is so, then PB_UP is the first acoustic computer music folk band: The laptop is their only instrument' (PowerBooks_UnPlugged 2015). The ensemble, started by Alberto de Campo, Echo Ho, Hannes Hölzl and Jankees van Kampen with input from Renate Weiser and Julian Rohrerhuber, gets its name from the practice of using the built-in speakers of the laptop as their playback system. Using this mobility to their advantage, the ensemble members typically distribute themselves amongst the audience during a concert, thus providing an inherently spatialised sound experience. Music is generated by live coding algorithms, which create the sound synthesis and executing them to improvise musical structures.

A software library called *Republic* was developed for PowerBooks_UnPlugged, to enable collaborative and distributed code-based music performance over a wireless network; a practice they prefer to call 'just in time programming', where they (re)write programs while they are already running. As described by one of its developers:

"Republic is an extension library for the SuperCollider language designed to allow highly collaborative forms of live coding. In a symmetrical network of participants, everyone can write and evaluate scripts which generate musical processes that can distribute sounds across all machines" (de Campo 2013, 22).

As well as sending music between machines, the system allows for chat communications that facilitate coordination amongst the ensemble. Performers are literally checking their 'email' on stage! The environment is designed to be deeply collaborative. 'The implicit working model is as democratic and symmetrical as the spatial disposition of the music: everyone can make sounds on her own laptop as well as (simultaneously or sequentially) on everyone else's' (Rohrerhuber et al. 2005).

The leaders of this ensemble are very clear that their focus is on 'improvising with algorithms', and their claim that this practice is a kind of 'folk' music for computers, has echoes of the composer Iannis Xenakis' search for authentic characteristics of digital music in his algorithmic formalisations (Xenakis 1992). Writing about their ensemble, members suggest that

"... a public improvisation with algorithms is no less plausible than experimenting with sounding objects on stage, and the numerous live coding approaches have led to an interesting variety of performances. Here, it is an ever changing dynamics of reprogrammed microcompositions that make up the improvisational situation, playing with the double time structure of processual change and change of the process" (Rohrerhuber et al. 2007).

The educational affordances of laptop ensembles, such as PowerBooks_UnPlugged, include sharing the workload and risk amongst the participants. That is, because only a small fraction of the extraordinary sonic potential of each laptop needs to be harnessed by each performer, this allows performers to focus on manageable fragments of code and on the generation and

manipulation of a restricted algorithmic process. This provides an achievable entry point for new musicians and, even in the case of a crash or error on one computer the distributed nature of the work means this has only a minor impact on the performance. The level of ensemble integration in a wirelessly shared computing environment, like Republic, can encourage a deep level of ensemble integration. 'Since instruments and control algorithms are shared, there's no real owner anymore; the creators are discrete musical entities only if they choose to be, ideas belong to everyone' (PowerBooks_UnPlugged 2015). The inherently distributed nature of the Republic software used by PowerBooks_UnPlugged enables performers to exploit these features maximally, as educational affordances, thus reinforcing ensemble performance skills.

Sound Thinking

The obviously interdisciplinary nature of computer music studies lends itself to collaborative courses between the arts and sciences. One such course is *Sound Thinking*, offered at the University of Massachusetts Lowell, which will be discussed in this section. Another is the course *Computer Music on a Laptop: Composing, Performing, Interacting* taught at the College of Charleston, which is the topic of the next example. These courses exist through cooperation between teaching staff in music and computer science departments, and are particularly feasible in liberal arts educational contexts, where students are encouraged to take courses in both the arts and sciences—often computer music courses can count for credit in either or both areas.

The design and the teaching of the Sound Thinking course were led by Jesse Heines, Gena Greher and Alex Ruthmann. It was designed as part of a teaching initiative called Performamatics that was devised to attract students to computing by tapping into their inherent interest in the performing arts (Ruthmann et al. 2010).

After some experimentation with a variety of technical platforms, the Sound Thinking course settled on the use of the Scratch environment (<https://scratch.mit.edu/>), developed at the MIT Media Lab. Scratch is a hybrid of text and visual programming paradigms, designed for young learners (see Figure 3). It has a strong focus on interaction and media outcomes such as animation and games. Its dynamic, media rich environment suits the performative nature of musical activities. In the Sound Thinking course students develop various generative music algorithms and learn to manipulate these for variation during performances. The course also includes the integration of hardware controllers—the MaKey MaKey board, developed by Jay Silver and Eric Rosenbaum, and formerly the locally made IchiBoards—to facilitate real-time gestural control of musical parameters.

Figure 3. A code block from the Scratch environment.

Algorithmic thinking is fostered by having students design musical flowcharts for various analytical tasks and compositional challenges. Flowcharts underscore the structural elements of musical compositions as a way of connecting algorithmic and musical designs. Algorithms studied in this course include random walk melodies, iteration through pitch and rhythm lists, and transposition through the use of offsets to MIDI note numbers.

In a book based on their experiences in these courses, *Computational Thinking In Sound*, Greher and Heines stress the course's focus on analytical skills and computational thinking which, they suggest, are just as important to music as they are to computer science. They also emphasise the benefits of the interdisciplinary mix of students that helps learners to break out of familiar habits of thinking and acting (Greher and Heines 2014).

JythonMusic

The College of Charleston, in South Carolina, is the site of another successful collaboration between music and computer science educators in the use of algorithmic music as the basis for interdisciplinary curriculum design. Courses in music fundamentals and introductory programming have been combined into new courses that include a variety of tasks including composition, interactive music systems, and ensemble performance. Two courses were designed and co-taught by Bill Manaris, Blake Stevens and Yiorgos Vassilandonakis: *Introduction to Computer Music and Aesthetics: Programming Music*, *Performing Computers* and *Computer Music on a Laptop: Composing, Performing, Interacting*. These courses were intended to ‘synthesize creativity in the arts with the ability to model and automate processes in code’ (Manaris et al. 2016: 44). The Introduction course has no prerequisites but students usually enter with some background in music performance or after having taken music classes at school; there is no expectation of prior computer programming experience. The Computer Music course is an honours-level offering and focuses on principles of music composition and computer programming for developing interactive computer music applications.

The JythonMusic programming environment was developed alongside the curriculum to support these courses (<http://jythonmusic.org>). It uses the Jython Environment for Music (JEM) editor for writing and evaluating code (see Figure 4). JythonMusic provides libraries for music data, audio playback, image manipulation, building graphical user interfaces (GUIs), and for connecting to external MIDI and OSC devices (Manaris and Brown 2014). A ‘musical’ data structure, inherited from jMusic, is used for representing musical scores. It includes classes for Note, Phrase, Part, and Score, as well as providing classes to represent audio material. Playback of score data is via internal or external virtual synthesizers. This capability fits well into, and supports, existing music curricula.

Figure 4. The Jython Environment for Music (JEM) editor.

Activities in their curriculum include the study of temporal musical structures, algorithmic processes, soundscape design, graphical user interfaces, programming design patterns, data types, language syntax and semantics, musical terminology, and characteristics of musical style. Students are required to produce various musical artefacts including standalone and interactive music software, solo and group compositions, and interactive performances. Project-based pedagogy is used as a vehicle to promote the integrated development of computing and musical skills and understandings.

Reflecting on their teaching of several iterations of these courses, staff say that a key benefit for students is that 'this coupling [of music and computing] leads to an increase in active and creative learning experiences, as each student gains proficiency in realizing and expressing musical ideas on a common instrument' (Manaris et al. in press). In addition to these participant observations, student surveys have been conducted. Results show that most are comfortable in regarding coding as a valuable medium for musical thinking and a legitimate form of musical composition. There is also strong support for music as an effective context for learning how to program. The surveyed cohort was less convinced that code-based music performances were capable of achieving musical outcomes comparable with more traditional methods of music making. As to their likelihood of continuing with music programming beyond the courses, the Introductory class responded very positively, while students in the Honours course were more polarised in their responses (Manaris et al. 2016: 35).

Overall, this example reveals a comprehensive engagement with the use of musical algorithms as the basis for an interdisciplinary offering in computer music education. After a sustained effort over five years, the courses are more refined and the faculty involved have produced rich resources, including a development environment, a book full of examples, and educational research that reflects their experiences.

Sonic Pi

Sonic Pi (<http://sonic-pi.net/>) is an open source music programming environment, created by Samuel Aaron and a team of voluntary developers. Sonic Pi's focus is on live coding and it is designed to be as easy as possible for beginners. From the outset there was a clear emphasis on learning pathways; and there are associated 'schemes of work' for music lessons. Sonic Pi has evolved through collaboration between musicians, academics and educators interested in helping school children to learn programming by creating music. As with many of the previous examples, interdisciplinary teams have enriched the development of Sonic Pi, and experiences from workshops and classes have guided its evolution. The platform was developed alongside pedagogical strategies and teaching materials. It comes with a section of example projects and lesson plans for teachers. The multifaceted motivations of those involved include: empowering children to access computing skills, promoting new musical habits and skills, and promoting creative partnerships between schools and communities (Aaron et al. 2016).

The name Sonic Pi results from a close link with, and financial support from, the Raspberry Pi Foundation who produce a low-cost computing platform on which Sonic Pi can run; there are also versions for OS X and Windows (see Figure 5). The partnership with the Raspberry Pi Foundation was facilitated by common connections at the University of Cambridge Computer Laboratory. The Sonic Pi environment includes an Interactive Development Environment (IDE) (see Figure 5) and uses a customised version of the Ruby language as an interface to a SuperCollider server that provides the core audio capabilities.

Figure 5. The Sonic Pi interface on Mac OS.

Sonic Pi examples and materials focus on compositional structures described in its scripting code and make use of prebuilt instruments and audio samples for sound output. Algorithmic processes explored in the examples include: stochastic choice, repetition and iteration, data slicing and recombination, and isorhythms. The stylistic outcomes range from ambient to hip-hop.

In keeping with the Sonic Pi project's focus on teaching programming, it is not surprising that the teaching materials are organised around a series of coding topics such as syntax, debugging, data structures and so on. For the music educator, the examples provide a more convenient launch pad from which musical principles can be unpacked. Music educators might also be interested in exploring the Sonic Pi Live Coding Toolkit (<http://www.sonicpiliveandcoding.com/>), which provides support materials for music pedagogy with artistic examples in the form of 'Pop Pi' videos and a range of suggested activities focused on a music curriculum.

The pedagogical approaches developed during classes and workshops emphasise role-play and group work as ways of augmenting coding and listening. This is in keeping with teaching strategies employed in computational thinking courses. More aligned with arts pedagogy and constructivist approaches is the Sonic Pi project's emphasis on participatory culture, experimentation and open-ended creativity, rather than on individual work and tasks oriented to predetermined outcomes (Aaron et al. 2016).

Conclusion

Algorithmic music processes have not been an everyday component of music education curricula. However, there is a growing number of tools and teaching materials that, in time,

should bring about change in this direction. Musical techniques, theories and methods have long been studied but their formalisation as algorithms and their articulation in computer programming languages are relatively new developments.

Algorithmic thinking privileges abstraction and generality—important concepts in both computer science and music composition—and therefore it has an important place in education in these fields. The leverage provided by computational automation means that algorithmic processes have a significant role to play in music production; their accessibility to educators and their prominence in music educational circles, however, are still developing.

This chapter has explored many of the issues involved in engaging with algorithmic representations of music for educational purposes. It has also provided an overview of several examples which show how the programming of music algorithms is being approached as a technique for assisting and motivating students to learn music and computer programming. Having identified some of the central opportunities and concerns of algorithmic music in music education, and after reviewing innovative examples, it seems clear that musical algorithms and coding skills are useful conceptual tools that can assist in the development of musical intelligence but have yet to be fully embraced by the music education community.

References

- Aaron, Samuel, Alan F. Blackwell, and Pamela Bernard. 2016. "The Development of Sonic Pi and Its Use in Educational Partnerships: Co-Creating Pedagogies for Learning Computer Programming." *Journal of Music, Technology & Education* 9 (1): 75-94.
- Abelson, Harold, and Gerald Jay Sussman. 1996. *Structure and Interpretation of Computer Programs*. 2nd Edition. Cambridge, MA: The MIT Press.
- Adkins, Barbara, Marie Knox, Steve Dillon, Andrew R. Brown, and Jennifer Summerville. 2012. "Digital Technologies and Musical Participation for People with Intellectual Disabilities." *New Media and Society* 14 (7): 501–18.
- Bamberger, Jeanne. 1979. *Logo Music Projects: Experiments in Musical Perception and Design*. MIT Artificial Intelligence Laboratory.
- Bamberger, Jeanne. 2003. "The Development of Intuitive Musical Understanding: A Natural Experiment." *Psychology of Music* 31 (1): 7–36.
- Code Club World Ltd. 2015. <http://codeclubworld.org/> Accessed April 2015.
- Cope, David. 2001. *New Directions in Music*, 7th Edition. Prospect Heights: Waveland Press.
- Collins, Nick, this volume – full reference required.
- Dave Conservatoire. 2016. *Introduction to Sonic Pi*. Accessed July 2016.
<http://www.daveconservatoire.org/course/introduction-to-sonic-pi>
- De Campo, Alberto. 2013. *Republic: Collaborative Live Coding 2003–2013*. Collaboration and learning through live coding Vol. 3 Issue 9. Dagstuhl Reports. Germany: Schloss Dagstuhl.

- Denning, Peter J. 2009. "The Profession of IT Beyond Computational Thinking." *Communications of the ACM* 52 (6): 28–30.
- Dillon, Steve. 2007. *Music, Meaning and Transformation: Meaningful Music Making for Life*. Cambridge: Cambridge Scholars Publishing.
- diSessa, Andrea. 2000. *Changing Minds: Computers, Learning, and Literacy*. Cambridge, MA: The MIT Press.
- Edwards, Michael. 2011. "Algorithmic composition: computational thinking in music." *Communications of the ACM*, 54(7), 58–67.
- Freeman, Jason A. 2014. *Survey of Music Technology*. Georgia Institute of Technology: <https://www.coursera.org/course/musictech>, accessed April 2015.
- Green, Lucy. 2002. *How Popular Musicians Learn: A Way Ahead for Music Education*. London: Ashgate.
- Greher, Gena R, and Jesse M Heines. 2014. *Computational Thinking In Sound*. New York: Oxford University Press.
- Griffiths, Dave. 2007. "Fluxus". In *Collaboration and learning through live coding, Report from Dagstuhl Seminar*. Vol. 13382, pp. 149-150. Schloss Dagstuhl
- Guzdial, Mark. 2008. "Education Paving the Way for Computational Thinking." *Communications of the ACM* 51 (8): 25–27.
- Guzdial, Mark, and Barbara Ericson. 2010. *Introduction to Computing and Programming in Python: A Multimedia Approach*. Upper Saddle River, NJ: Pearson.
- Jorgensen, Estelle R. 2003. *Transforming Music Education*. Bloomington: Indiana University Press.
- Kapur, Ajay. 2013. *Introduction to Programming for Musicians and Digital Artists*. CalArts: <https://www.coursera.org/course/chuck101>, accessed April 2015.
- Levtov, Yuli. This Volume – **needs full reference when available**.
- Levenson, Thomas. 1994. *Measure for Measure: A Musical History of Science*. New York: Touchstone.
- Luckin, Rosemary. 2008. "The Learner Centric Ecology of Resources: A Framework for Using Technology to Scaffold Learning." *Computers & Education* 50 (2): 449–62.
- Matthews, Charles, this volume – **full reference required when available**.
- Manaris, Bill, and Andrew R Brown. 2014. *Making Music with Computers: Creative Programming in Python*. Chapman & Hall/CRC Textbooks in Computing. Parkway, NW: CRC Press.

This is a draft of a chapter/article that has been accepted for publication by Oxford University Press in the Oxford Handbook of Algorithmic Music, edited by Alex McLean and Roger Dean and published in 2018.
Brown, Andrew R. 2018. "Algorithms and Computation in Music Education." In *The Oxford Handbook of Algorithmic Music*, edited by Alex McLean and Roger T. Dean, 583–602. New York: Oxford University Press.

Manaris, Bill, Andrew R. Brown, and Blake Stevens. 2016. "JythonMusic: An Environment for Teaching Algorithmic Music Composition, Dynamic Coding, and Musical Performativity." *Journal of Music, Technology and Education* 9 (1): 33-56.

Maeda, John. 2004. *Creative Code*. London: Thames & Hudson.

Nierhaus, Gerhard. 2010. *Algorithmic Composition: Paradigms for Automated Music Generation*. New York: Springer.

Ogborn, David, this volume – a full reference is required here when available.

Papert, Seymour. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books.

PowerBooks_UnPlugged. 2015. "PowerPlugs_UnBooked or: PowerBooks_UnPlugged." <http://pbup.net/s/>. Accessed April 2015.

Reas, Casey, and Benjamin Fry. 2003. "Processing: A Learning Environment for Creating Interactive Web Graphics." In *International Conference on Computer Graphics and Interactive Techniques*, 1–1. San Diego: ACM SIGGRAPH.

Resnick, Mitchel. 1994. *Turtles, Termites, and Traffic Jams*. Cambridge, MA: The MIT Press.

Roberts, Charlie. 2013. "Gibber." <http://www.charlie-roberts.com/gibber/>. Accessed April 2015.

Roberts, Charlie. This Volume – needs full reference when available.

Rohrhuber, Julian, Alberto de Campo, and Renate Wieser. 2005. "Algorithms Today: Notes on Language Design for Just in Time Programming." In *Proceedings of the International Computer Music Conference*, n.p. Barcelona: ICMA. <http://quod.lib.umich.edu/i/icmc>.

Rohrhuber, Julian, Alberto de Campo, Renate Wieser, Jan-Kees van Kampen, Echo Ho, and Hannes Hölzl. 2007. "Purloined Letters and Distributed Persons." *Music in the Global Village*. http://globalvillagemusic.net/2007/wp-content/uploads/pbup_paper.pdf. Accessed April 2015.

Ruthmann, Alex, Jesse M Heines, Gena R Greher, Paul Laidler, and Charles Saulters. 2010. "Teaching Computational Thinking through Musical Live Coding in Scratch." In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 351–55. Milwaukee: ACM.

Sylvan, Elizabeth. 2006. "Who Knows Whom in a Virtual Learning Network? Applying Social Network Analysis to Communities of Learners at the Computer Clubhouse." In *Proceedings of the 7th International Conference on Learning Sciences*, 758–64. Bloomington, IN: International Society of the Learning Sciences.

Swanwick, Keith. 1994. *Musical Knowledge: Intuition, Analysis, and Musical Education*. London: Routledge.

This is a draft of a chapter/article that has been accepted for publication by Oxford University Press in the Oxford Handbook of Algorithmic Music, edited by Alex McLean and Roger Dean and published in 2018.

Brown, Andrew R. 2018. "Algorithms and Computation in Music Education." In *The Oxford Handbook of Algorithmic Music*, edited by Alex McLean and Roger T. Dean, 583–602. New York: Oxford University Press.

Taube, Heinrich K. 2012. "Computation and Music." In *Sound Musicianship: Understanding the Crafts of Music*, edited by Andrew R. Brown, 40–51. Newcastle upon Tyne: Cambridge Scholars Publishing.

Taube, Heinrich. 2004. *Notes from the Metalevel: Introduction to Algorithmic Music Composition*. London: Taylor & Francis.

Wing, Jeannette M. 2008. "Computational Thinking and Thinking about Computing." *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366 (1881): 3717–25.

Xenakis, Iannis. 1992. *Formalized Music: Thought and Mathematics in Music*. Stuyvesant NY: Pendragon Press.