Jive: A Generative, Interactive, Virtual, Evolutionary Music System

Jianhua Shao[1], James McDermott[2], Michael O'Neill[2], and Anthony Brabazon[2]

[1] University of Nottingham dustin.shaojianhua@gmail.com
[2] University College Dublin jamesmichaelmcdermott@gmail.com m.oneill@ucd.ie anthony.brabazon@ucd.ie

Abstract. A novel paradigm and system for interactive generative music are described. Families of musical pieces are represented as functions of a time variable and several variables under user control. Composition/performance proceeds in the following two stages. Interactive grammatical evolution is used to represent, explore, and optimise the possible functions. The computer mouse or a Wii-controller can be used for real-time interaction with the generative process. We present rationale for design decisions and several pieces of example music.

Key words: Generative music, evolutionary computation, grammatical evolution, interaction

1 Introduction

Generative music is music which is not specified by a score, but by an algorithm, a set of rules, a set of processes, a mapping from randomness, or some other such method. Collins [1] provides a good introduction, quoting a definition of generative art as art that is "generated, at least in part, by some process that is not under the artist's direct control" [2]. Of course this requires a definition of "direct". Collins also quotes Sol LeWitt: "the idea becomes a machine that makes the art". This brings to mind a famous remark made in the context of meta-programming: "I'd rather write programs that write programs than write programs." (Richard Sites). Generative art is "meta", in the same sense: the artist creates not a single instance of the work, but instructions with the potential to create a family of instances. Meta-programming is also an example of the observation that constructive laziness is a characteristic of good programmers. In this spirit, we like the implicit description given by Brian Eno, who has been responsible for popularising both the term "generative art" and the musical genre: "[...] I've always been lazy, I guess. So I've always wanted to set things in motion that would produce far more than I had predicted." [3].

Collins [1] also mentions an important distinction, that between interactive and non-interactive generative art. In the latter, the artist cannot intervene after the generative process has begun. Interactive generative music allows intervention and "performance": at one extreme, a musical piece such as Queen's

Brighton Rock might be seen as interactive generative music, where the interaction has been increased to a full instrumental performance and the generative aspect reduced to an extreme echo effect.

Our focus in this paper is a form of interactive generative music where material is created through more typical generative processes—musical processes embodied as algorithms and equations—but interaction by direct manipulation of some of the equations' parameters is possible, via either the mouse or the popular, intuitive Nintendo Wii Remote. Since our system is intended to be usable by anyone with no computer/mathematical background or training, we adopt a point of view characterised by the term "hidden variables". Parameters are not intended to be explicitly understood by the performer. Rather, the performer gradually and implicitly learns their effects in different contexts.

The search for algorithms and equations leading to interesting generative music with viable interaction possibilities is a difficult task even for users with computer experience. The space of possible equations and algorithms is large. We therefore require a tool for navigating it, and we choose interactive grammatical evolution (GE), a form of interactive evolutionary computation (IEC) which has been successful in previous applications [4–8]. Our system, called "Jive" (for "generative, interactive, virtual, evolutionary") thus allows two levels of composition/performance. First, the creation of the generative piece itself is a compositional process done through IEC. It fixes many aspects of the "family of instances". Secondly, performance of a particular instance is done by live control of hidden variables. During the IEC process, many short experimental instantiations will be created as the user comes to grips with the possibilities presented by the evolving population. This two-phase process, using IEC to create interactive generative music, is the central novel contribution of this paper.

The remainder of this paper is laid out as follows. Previous work is reviewed in Sect. 2. The Jive system is described, with motivation and examples for its design decisions, in Sect. 3. Results obtained using this system, and refinements based on their success and failure, are given in Sect. 4. The final sections contain discussion, conclusions and future work.

2 Previous Work

Evolutionary approaches to music generation are well-known [4, 5]. Generative processes such as L-systems have been explored both within and without the evolutionary context [9]. Magnus' "Evolutionary Musique Concrète" [10] and many others have used evolutionary dynamics as the primary means of driving the development of music over time. Others have used non-interactive EC with computational fitness functions [6]. These differ from the approach adopted here, where we see interactive EC as a tool, and the generative aspect of the music could exist independently of EC.

The *Genophone* system [7] has some of the same aims as that explored in the present work, in that both *performance mappings* and material are created. There are two major differences. *Genophone* operates at the level of sound

synthesis parameters, whereas our focus is on score-level generation. Also, our representation, using context-free grammars, is entirely different.

However the most direct source of inspiration for this work is the NEAT-based "compositional pattern-producing network" approach [11, 8]. Complex networks of functional relationships map input variables to plausible, realistic output music (or graphical art, etc.). The functional networks are created using IEC. Although the input variables are derived from pre-written input music (the aim is to automatically produce rhythm tracks to accompany existing music), the approach has more general potential to map any input parameters to output music. This is the point we take up.

The commercial system *Noatikl*, which is descended from *Koan*, used to create Eno's seminal *Generative Music 1*, allows user interaction with generative pieces according to a non-evolutionary paradigm.

Generative grammars, like the context-free grammar used in our grammatical evolution approach, have been extensively used for both the analysis and generation of music, for example by Lerdahl and Jackendoff [12]. However, we wish to draw an important distinction here. In our work, we believe for the first time, the generative grammar is used to create code—specifically a set of arithmetic and boolean functions, which drive the generative process. It is not used to create musical material directly, and so our work has no direct bearing on grammatical theories of music.

3 The Basic Jive System

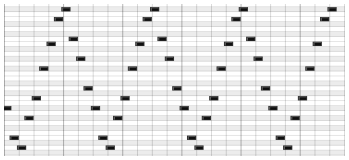
The Jive system consists of four components: generative, interactive, virtual, and evolutionary. They are described in the following four sections.

3.1 Generative

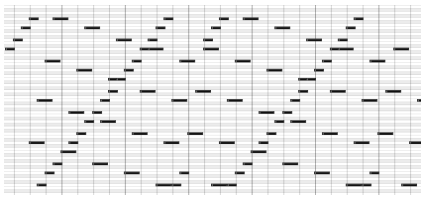
Fundamentally, Jive is a generative music system in which music is a function of time, and time is seen as a discrete variable. In the simplest possible case, a function such as $f(t) = 40 + 7\sin(2\pi t)$ or $g(t) = t \bmod 60$ will create a piece of music, albeit a very boring one. Here, the output of functions f and g is a number, which we round (if necessary) to an integer and interpret as a MIDI note number. We make one immediate improvement, however: since we wish to minimise the random feel associated with much generated music, we will map the integer to a diatonic scale.

There are several routes towards creating more interesting music. First, we require the ability to play *any* succession of notes. It is well-known that a combination of sinusoidal functions is sufficient to represent any periodic function of one variable (and since a piece of music will be finite in length, the periodicity requirement is unnecessary). This universality is the property we require. Another possibility, which leads more immediately to musical results, is to use combinations and variations of a linear function like $h(t) = a + q(t \bmod p)$. By varying the values of a (a *pitch offset*), q (a *scaling* factor), and p (a *periodicity*

parameter), we can produce a function of one variable, time, which produces ascending and descending scales, and in general piecewise-linear sequences, as depicted in Fig. 1(a)[3].



(a) Simple piecewise-linear results obtainable using a single h function



(b) A single voice can become arbitrarily complex by summing multiple h functions.

Fig. 1. Examples demonstrating output of a single voice.

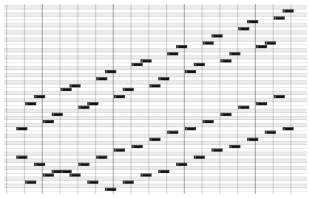
The important role played by the mod operator is to provide periodicity. A similar role might have been played by a sin function as noted above. A single instance of the h function chunks time into periods, giving our generative piece repetitive pattern. We can also sum multiple instances of the function, with varying values for a, q, and p, to obtain a function $h(t) = \sum_i h_i$, again of one variable, time, which can produce any desired sequence of single notes[4]. In order to keep things rhythmically coherent, we will constrain p to take on values of the form $p = p_1$ or $p = p_1 p_2$, where p_1 and p_2 are small integers (2, 3, 4, or 6), and their values are fixed for a given piece of music. They function as the primary and secondary *rhythmic characteristics* of the piece. The summed h function will have periodicity equal to the lowest common multiple (LCM) of its component periodicities: the LCM is constrained by this scheme to be a relatively low value. The types of results obtainable using this scheme are depicted in Fig. 1(b).

The next step is to add a secondary pattern at a longer time scale. We achieve this using the *quotient* function. We alter our summed function to allow expressions of the form $h_i(t) = (t \text{ quot } a) + q(t \bmod p)$. The quot function performs integer division. During the time-steps 4-7, the expression t quot 4 has a constant value, 1, which is used as an offset. This causes any simple pattern created by q, t and mod to be repeated at different offsets, giving a harmonic feel, as in Fig. 2(a).

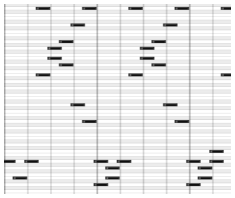
Multiple voices are not difficult to achieve: we can simply create new functions $h' = \sum h'_i, h'' = \sum h''_i$, and so on. For now we stick to three such functions. Each calculates pitches using independent parameter values, so a higher degree

[3] These clips demonstrating successive levels of development, together with software, example grammars, and four demo pieces, are available at http://sites.google.com/site/odcsssjian2009/.

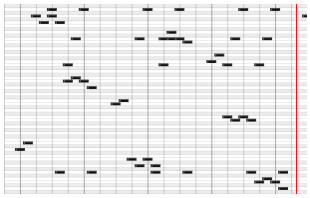
[4] This is like genetic programming-style symbolic regression, for music.



(a) The *quotient* function gives a feeling of harmonic movement.



(b) Multiple independent voices.



(c) Omitting some notes gives a first approximation of phrases.

Fig. 2. Simple examples of harmony and phrasing.

of complexity in pitch-movement can now arise. There is one exception: periodicity is still constrained by the primary and secondary rhythmic characteristic parameters. Some examples are given in Fig. 2(b).

We add the possibility of rests, rather than a constant succession of notes. We add an independent boolean function to each voice, which calculates a true or false value at each note indicating whether to play the pitch calculated for that voice by its numerical function, or to remain silent. The boolean function is created, for now, using `<`, `<=`, `==`, and other comparisons of t against arithmetic expressions similar to those used for pitch calculation. This allows much more interesting phrase-structures to emerge. The music being generated is still very simple but is suddenly beginning to sound like music. Some examples are depicted in Fig. 2(c).

Finally, for now, we can make the form of our equations entirely open-ended. Instead of using the fixed function h with varying numerical parameters, as described earlier, we can write a context-free grammar which creates arbitrary functional expressions in our input variables, using other periodic functions such as sin() and cos() combined with multiplication, addition, subtraction and (protected) division. It quickly becomes impossible to predict the style of an individual from inspection of its code, but this approach has the advantage of being more open-ended. A good generative system will sometimes surprise its creator, and this is more likely to occur using an open-ended representation. It is certainly capable of producing compositions which the authors of this paper could not have written by hand.

3.2 Interactive

Since the system as described in the previous section creates material as a function of time rather than from an explicit score, it may be regarded as generative. However it is capable only of producing simple periodic pieces. A key aim in this research is to allow the user/performer to interact with the generative music as it plays. This will allow the material to change and develop over time. Hence, we add to our system some continuous-valued variables representing user input.

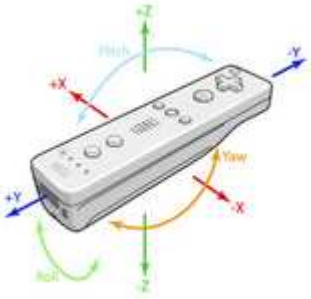
These are then available to be incorporated into the numerical expressions for pitch and boolean expression for note presence/absence.

We provide multiple variables, which will be controlled by the user as described in the next section, and we allow some flexibility in the way they are used. In general, they can be used as offsets or scaling parameters to existing parameters in our equations. Each input variable may be used more than once in our various equations, but there is no requirement that every variable be used at all. This indirect, optional, and multiple usage of input variables we refer to as "hidden variables", as discussed in more detail in Sect. 5.

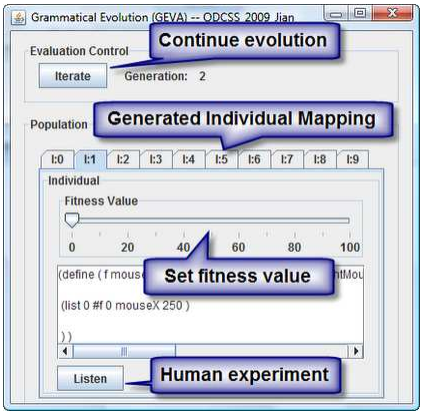
3.3 Virtual

The continuous user-input variables referred to in the previous section are the user/performer's main means of interaction with a playing piece (it is also possible to change tempo manually, but this is not of interest here). Although the boundary between sequencer, performance system, and instrument is blurred by the system we have described, we refer to Jive as a *virtual instrument*. This term is typically used to mean playable musical instruments which are "disembodied" or implemented purely in software or electronics.

The user/performer can perform with the Jive system in two ways. The mouse is a simple method: it provides X and Y values. Although the system has been programmed to read multiple mouse buttons also, these are not used in any current configurations. The Nintendo Wii remote control, also known as the "wiimote", is a more sophisticated option. It can provide either absolute position or accelerometer data in three dimensions, as well as multiple buttons. Again, we currently use only a subset: the absolute X, Y and Z values. The wiimote, shown in Fig. 3(a), does not require a GUI. It is interfaced to our software using the WiiRemoteJ library (http://www.world-of-cha0s.hostrocket.com/WiiRemoteJ/).



(a) The Nintendo Wii Remote



(b) GUI used for auditioning and fitness evaluation

Both controllers are sufficient for their intended purpose here. Small movements tend to lead to small changes in output; larger movements can lead to entirely different behaviours. Well-timed movements are required, but great dexterity is not. More interesting interaction possibilities can be created by sending not only current 2D or 3D position data, but also a buffer of recent positions, as input variables to the equations. In the case of the wiimote, for example, a buffer of size 3 now gives us 9 continuous input variables in addition to time, and the pitch equations and boolean note presence/absence functions can use any or all of them. One advantage is that smoother changes occur—however there is a trade-off with a loss of fine control, since the sound being played at each instant is now dependent on previous controller movements. Sometimes the periodic patterns that occur, using the buffer, seem less rigid and more natural. It allows for potential "emergent gesture recognition", since in principle relative movement as well as absolute position are now available to the system.

3.4 Evolutionary

The system as described so far is complete, in that it can be used to create and to perform pieces of music. It is not user-friendly: most musicians, if they are human, are not capable of performing symbolic regression in their heads while composing. Instead, we now introduce the last component of the system, interactive evolutionary computation (IEC). This algorithm works in the same way as typical EC, but allows the user to specify individuals' fitness values, or to perform selection directly. Because it generates multiple individuals in an iterative process, with the user required only to assess their quality, it avoids the need for the user to write equations directly when creating a piece.

The IEC GUI used for generation, auditioning and selection of pieces, and iteration of the algorithm, is shown in Fig. 3(b). It remains to describe the representation we have chosen for our individuals.

Grammatical evolution (GE) is a proven technique for representing code in diverse languages with arbitrary constraints on its form [13]. The language syntax and the allowable or desirable syntactic forms (such as our equations) are specified in a context-free grammar in Backus-Naur Form. An individual's genome is an integer array, which specifies the grammar productions to be chosen during the derivation process.

In our work, GE is implemented using the freely-available GEVA software [14], written in Java. GEVA was configured to use a population size of 10, an unlimited number of generations, a one-point crossover rate of 0.7, an int-flip mutation rate of 0.02, and generational replacement. Our equations and boolean functions, together with some boilerplate code for dealing with input variables and time, were specified as BNF grammars producing programs in the JScheme language [15]. The grammars we have used are available for download: see Sect. 6. The jMusic API is used to take the values returned by execution of JScheme individuals and translate them into MIDI.

Although any given version of our equations might be represented as a linear, real-valued genome—one gene per parameter—we have found that the GE

approach, using a BNF grammar, gives much greater flexibility. The process of altering the *form* of an equation is much easier with a grammar, compared to re-writing code. Grammars can also be entirely *open-ended*, allowing (for example) a sum of multiple expressions h'_i, or an XOR-combination of boolean note presence/absence—and the number of such expressions may be unknown in advance and left for evolution to determine. This is much more difficult or impossible using a GA-style representation.

4 Results and Refinements

The basic system as described so far is already capable of producing some music sufficiently interesting to be worth describing.

The system is clearly lacking in the area of rhythm. Phrases are essentially created by knocking notes out, i.e. switching on a "rest" flag for one or more notes, leaving gaps between sub-sequences which the ear then interprets as isolated phrases. In the demo pieces RD_0 and RD_2, we attempt to work against this shortcoming. An open-ended grammar makes a good deal of complexity available during the performance phase. Multiple basic behaviours were available in the mouse's (x, y) plane, for example an "ascending" behaviour, partly controlled by relative position; a "stop" or steady-state behaviour; and a dense periodic pattern. Composition in this case consisted not only of choosing which behaviour to switch to, but when. It was possible to take advantage of this to create higher-level structure in the pieces. It was also possible to use a voice creating sparse material, whose pitch was under direct control, to act as a melody. This put the continuous material in the background, to some extent overcoming the "continuous stream" feel.

The demo piece ML_0 uses the "memory" facility and is very complex, chaotic at times. The piece was evolved through approximately 22 generations.

In the final demo piece, MD_0, we have added two new features. A voice can now choose to play a chord, chosen from a small selection of major and minor triads and sevenths. A drum grammar was also created and used in Jive to produce a looped rhythm track. Note that all demo pieces have been rendered in an external program using manually-chosen synthesizers and effects.

5 Discussion

Generative music at its best brings out the abstract patterns which underlie (but do not solely *constitute*) many (but not *all*) types of music. The "hidden variables" approach adopted here has some satisfying results in this context. A change in a single hidden variable may have multiple effects. This imposes a type of large-scale coherence on the output—multiple voices may react simultaneously to a change in a user parameter, for example. This can lead to a re-interpretation of non-generative music. When several instruments in an orchestral piece crescendo and then switch to a new section, one could interpret the multiple voices as manifesting the effects of shared, hidden variables.

An interesting aspect of generative music is the blurry boundaries it creates between composer, performer, and listener. This is augmented in our work by the explicitly interactive element. The user of the Jive system plays roles including programmer (editing the grammar to fix a different rhythm), critic and composer (auditioning and selecting generated pieces during interactive evolution), and finally composer, conductor and performer (interacting with a fixed piece). We have occasionally used terms like "user/conductor" and "user/performer" to emphasise this blurriness.

Sometimes the control available to the user/performer seems very crude. While performing, it is not (in general) possible to insert or delete arbitrary notes. This apparent drawback has two, perhaps unexpected advantages. Firstly, since the system generates material continuously, with precise timing and no "wrong" notes, the user/conductor is freed from low-level details. The mouse and to a lesser extent the wiimote are, after all, inadequate controllers for the type of dextrous performance required by typical musical performance.

Secondly, the user/conductor gains a higher-level type of control despite the lack of low-level detail. As discussed in Sect. 4, the user/composer has control not only of which behaviour to switch to, but also when. He or she is not only performing the gestures which correspond to desired sonic results, but is reacting to the current musical context created by the ongoing algorithm and by his/her previous actions. This ability allows the censoring of undesired sections and the creation of complex musical syntax, such as call-and-response patterns among the several behaviours. The user's control of timing is sufficiently fine to allow the behaviours to work together in interesting ways.

6 Conclusions and Future Work

In this paper, we have presented the Jive system, a novel system and paradigm for interactive generative music. It uses IEC with aesthetic selection to create a generative process, and is a virtual instrument for real-time expression and performance. We have explained the design decisions and shown examples of the possible outputs. We consider that the styles of music possible with Jive, though limited, are interesting and characteristic.

Any virtues the system has arise from two sources. Firstly, our representation includes set of primitives which are not in themselves musical but function well as building-blocks for music. Secondly, we use a two-phase composition/performance process. The user/performer has some ability to overcome failings present in the composed piece, as discussed in the context of the demo pieces. The choosing of constraints and the exploration of what can be done despite them is characteristic of many art-forms. There is also the possibility of a third phase, in which the user edits the grammar in order to constrain the possible outputs. Of course this is only available to technically-oriented users.

The Jive system is not finished, and there are several possibilities for improvement. In our future work, our first priority is to address the system's shortcomings in the area of rhythm. We intend to add the possibility of more complex

rhythms with the addition of triplets, quintuplets, etc. More complexity in the phrasing and grouping of notes is also required. The interactive evaluation of multiple individuals per generation is a time-consuming task. There is some scope for automatic evaluation, for example by filtering individuals which give over-dense or inadequately varied material.

We believe that the generative, hidden-variable approach used here is a good way of making musical games, similar in spirit to popular "guitar karaoke" games but far more open-ended and expressive.

Acknowledgements

Jianhua Shao's work was funded by Science Foundation Ireland under the OD-CSSS scheme. James McDermott is funded by the Irish Research Council for Science, Engineering and Technology under the Empower scheme.

References

1. Collins, N.: The analysis of generative music programs. Organised Sound **13**(3) (2008) 237–248
2. Boden, M.: What is generative art? (October 2007) COGS seminar.
3. Eno, B.: Generative music. Online (June 1996) Lecture, San Francisco, USA.
4. Miranda, E.R., Biles, J.A., eds.: Evolutionary Computer Music. Springer (2007)
5. Bentley, P.J., Corne, D.W., eds.: Creative Evolutionary Systems. Morgan Kaufmann (2002)
6. Dahlstedt, P.: Autonomous evolution of complete piano pieces and performances. In: Proceedings of Music AL Workshop. (2007)
7. Mandelis, J., Husbands, P.: Genophone: Evolving sounds and integral performance parameter mappings. International Journal on Artificial Intelligence Tools **15**(4) (2006) 599–622
8. Hoover, A., Rosario, M., Stanley, K.: Scaffolding for interactively evolving novel drum tracks for existing songs. In: Proceedings of EvoWorkshops. Volume 4974 of LNCS., Springer (2008) 412
9. McCormack, J.: Evolutionary L-systems. In Hingston, P.F., Barone, L.C., Michalewicz, Z., Fogel, D.B., eds.: Design by Evolution: Advances in Evolutionary Design. Springer-Verlag (2008) 169–196
10. Magnus, C.: Evolutionary musique concrète. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A., eds.: Applications of Evolutionary Computing, Berlin, Springer-Verlag (2006)
11. Stanley, K.O.: Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines **8**(2) (2007) 131–162
12. Lerdahl, F., Jackendoff, R.: A Generative Theory of Tonal Music. MIT Press, Cambridge, MA (1983)
13. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers (2003)
14. O'Neill, M., Hemberg, E., Bartley, E., McDermott, J., Brabazon, A.: GEVA: Grammatical evolution in java. SIGEVOlution **3**(2) (2008) 17–22
15. Anderson, K., Hickey, T., Norvig, P.: JScheme and JScheme documentation. http://jscheme.sourceforge.net/jscheme/main.html Last accessed 30 October 2009.