Oliver Bown
Centre for Electronic Media Art
Clayton School of Information Technology
Monash University
Clayton 3800 Australia
ollie@icarus.nu

Experiments in Modular Design for the Creative Composition of Live Algorithms

Musical Autonomy and Live Algorithms

Live algorithms for music (truncated to "live algorithms" or abbreviated as LAMs) were proposed as a conceptual framework by Blackwell and Young (2004, 2005) as a way of analyzing the broad range of live music performance systems that exhibit autonomy or agency in some form or another. They are intended for real-time performance, usually with instrumental musicians, but also on their own or with other live algorithms.

The coining of the term "live algorithms" pulls focus away from the ideal of a virtual musician, towards an eclectic mix of dynamic and interactive behaviors, more generally categorized as *open dynamical systems*. Although designers of some of the most successful live algorithms were specifically concerned with the simulation of human musical behavior through virtual musicians, they were also involved, actively or inadvertently, in revealing novel ways in which a computer can act as an improvising partner (Rowe 1993, 2001; Biles 1994; Winkler 1998; Lewis 2000). Thus, although "virtual musician" has certain connotations and perhaps undesirable or presently unrealistic expectations, the term "live algorithms" leaves open a wide variety of behaviors, ranging from weakly to strongly musician-like, with modest implications of agency. (The algorithm is itself "live" rather than being "for live use" by a performer.)

Blackwell and Young proposed that researchers and composers interested in live algorithms could collaborate easily by considering the most elementary conceptual breakdown of an entire live algorithm system into basic high-level components: analysis (P) and synthesis (Q) elements with a patterning process (f) in between. As a physically embodied agent, a live algorithm is a computational system connected to the world, minimally, via sensors and actuators relevant to music. Typically this would be a microphone (via an analog-to-digital converter) and loudspeakers (via a digital-to-analog converter). Such embodiment pinpoints the potential for modularity, a mechanism that facilitates substitution: One live algorithm can be substituted for another in a performance context, and live algorithms can be exchanged with musicians in an improvising ensemble. Modularity is a crucial feature of modern design, particularly collaborative design, and facilitates creativity by providing the basis for combinatoric search through a space of possibilities. It is compelling to consider what modular approaches can do to address the technical, collaborative, and creative challenges of developing autonomous musical agents.

Blackwell and Young's *PfQ* model suggests an additional layer in which a modular strategy can facilitate a functional mechanism for substitution: the system f that is nested between audio analysis and audio synthesis components (which are themselves modular; see Figure 1). The f component can be thought of as an abstract behavioral system: the brain of the live algorithm. However, the interface between P and f, and between f and Q, is not specified, and there is no reason to believe that there exists a suitable interface for universal collaboration between developers of different modules. A LAM workshop and concert in 2009, organized by the author at Goldsmiths College and Café OTO in London, had this question of intercomponent modularity in mind, but only settled on the broadest notion of communication between elements, where different programmers could specify relevant network messages, to which their collaborators could respond by designing behaviors that conformed to these specific impromptu messages. A number of interesting dynamical systems models were presented at the workshop, such as swarms, recurrent neural networks, and simulations of self-organizing criticality systems such as sand piles and forest fires. Each of these dynamical system behaviors

Computer Music Journal, 35:3, pp. 73–85, Fall 2011

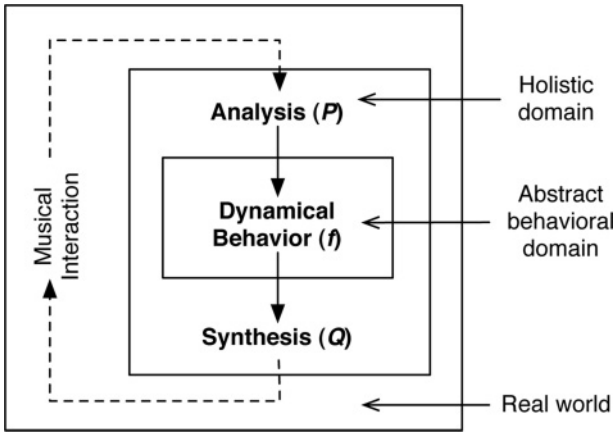


Figure 1. Levels of modularity in the live algorithms framework. The system is embodied and situated in the real world, interacting through sound. We can consider the system as a whole (holistic domain), and also by distinguishing its core modular subsystem (abstract behavioral domain) from its other modules (analysis and synthesis).

seemed to require different forms for input and output to be creatively embedded in a real-time musical agent, and the predominant tendency was for bricoleur-style (McLean and Wiggins 2010) creative design solutions. This tendency resulted in idiosyncratic systems lacking clear modularity and a clear understanding of interaction between elements.

This article presents examples of "composing" LAMs using a mixture of creative computing techniques. No attempt is made to find a general-purpose modular system as a viable solution for LAM design. Nevertheless, Blackwell and Young's *PfQ* model is a relevant way to approach the design of live algorithms, dividing a complex system into elements that can be developed in different ways, including using bricoleur-style hand-coding (which includes experimental substitution of different modular elements); target-based evolution of modules in simulated environments; interactive aesthetic evolution; learning and other forms of adaptation; and the tweaking of parameters in a traditional graphical user interface (GUI). Designers of LAMs may also combine precomposed score elements, audio samples, live processing effects, and more traditional GUI controls with the autonomous responsive behavior that is central to a live algorithm. Although these elements may be seen as generally detracting from the autonomy of the live algorithm, they are essential to contemporary digital musical contexts and are obvious approaches to endowing performance systems with musical knowledge. Such elements are unavoidable and should be accepted as openly as possible in order to lessen confusion about where autonomy lies in the resulting system.

Designing Creative Autonomy

Creative improvised music is a perplexing domain for artificial intelligence (AI) research. Given a relatively standardized musical improvisational form, such as blues, one can state certain relatively precise rules for how an interacting participant in a given role should act. However, musical styles rarely specify strictly what an improviser should do, as this would contradict the potential for improvisation and innovation.

The goals of AI in such an environment are unclear. If one focuses on performance in a specific, relatively strict style, it is possible to close in on a fairly well-defined search space of acceptable possibilities. For example, John Biles's celebrated GenJam system works in a standard jazz domain and is constrained by numerous compositional rules to produce acceptable jazz performance behavior (Biles 2001). Such domains may seem to be the most scientifically sensible choice for understanding musical creativity, and yet, because creativity requires novel variation as well as human evaluation in an appropriate context, the constraints of an established style do not make the problem any more tractable. At the other extreme, a creative LAM designer can invent an idiosyncratic musical context in which to study the use of interactive dynamical systems behavior, for example, by manipulating patterns of amplified background noise (Di Scipio 2003). This may result in more ambiguous musical challenges, but arguably provides a more compelling and relevant performance and evaluation context, one that can be actively examined in the real world of innovative musical practice.

Inherently, therefore, research into AI music systems involves an at least implicit distinction between a *prescribed* behavior and an *unprescribed* behavior. In the extreme case of a system designed to mimic a specific stylistic reference point, the prescribed element is strong, and the goal of building such a system becomes a technical challenge

(although one with a compositional nature). This is most evidently the case in David Cope's Experiments in Musical Intelligence (EMI, or the more human-sounding Emmy), in which he attempts to mimic various well-known historical styles while creating original compositions (Cope 1992). Cope's work poses a conundrum: How can the work be both original and true to the style being mimicked? Something has to give. We deride artists who work too closely in the style of others, and attribute little creativity to them. But although a more unprescribed approach may be more appealing, this increases the potential confusion between what the programmer is simply writing into the system and what the system is capable of originating.

Given the strong presence of an unprescribed element to musical AI, however, it is natural that researchers have proceeded by experimenting with intelligent and generative dynamical systems in musical contexts, without unambiguous scientific goals. Their challenge is part technical and part compositional.

Accordingly, attempts to produce AI musical systems have been largely attempted by people that take the role, at least in part, of creative musicians, who are suitably placed to test their ideas in real performance contexts. This may appear to further obscure the already-clouded issue of the agency of the system itself, but in the absence of a solid scientific grounding inherent in this field, it has emerged as an essential form of practice-based research.

With this in mind, a strong influence on the design of the systems presented here is the proposal that interactive musical systems should not need to demonstrate their interactivity; the interactive nature of the system can be obscure, mysterious, and opaque. Tests can demonstrate unambiguously that a system is responsive to the performer it is making music with by noting that the sound made by the performer has a causal effect on the material generated by the system. Such measures are insufficient to determine whether a successful musical interaction has been achieved. The quality of the interaction is determined by the evaluation of all involved: designer, performer, audience, or anyone studying the system. A performer may form his or her own conception of the system's behavior from two better-known modes of interaction: human-to-human interactions, and human-instrument interactions. But these are not essential reference points: Once a player or an audience has experienced new digital forms of interaction, they have new reference points. The live algorithms approach invites this reconceptualization.

Creative Modular Approaches and Behavioral Objects

Bown, Eldridge, and McCormack (2009) define *behavioral objects* as modular software elements that can be used in different contexts to achieve complex software behaviors, which will increasingly contribute to the performance capacities of creative behavioral systems such as live algorithms. An important aspect of behavioral objects is that, consisting merely of information, they can easily be shared among a community of musical users, along with audio samples, MIDI data, and other types of musical information. Exchangeable modular software elements such as Max/MSP objects, code libraries, and VST plug-ins are existing examples of behavioral objects, exhibiting modest autonomy but potentially immense complexity. Behavioral objects can also act as active agents in performance systems. The live algorithms framework advocates the use of any system that is capable of responsive, complex, dynamical behaviors as a generative musical mechanism, celebrating the rich musical potential of harnessing such a diverse set of systems.

In the spirit of a bricoleur-style programming approach, which is becoming increasingly entrenched as creative practice across the generative arts, different dynamical processes can also be merged and tweaked to fulfill immediate musical goals. A great challenge lies in managing the complexity of such systems and getting disparate behavioral elements to interact and work together to achieve musical goals.

For this author, an appealing goal is to seek frameworks for extensible and diverse musical behavioral possibilities that can be collaboratively developed and explored by a community of users.

Assayag, Block, and Chemillier (2006), for example, have developed a format for improvising *oracles*, different behavioral programs that can be loaded into a common performance framework. Assayag et al.'s oracles are derived from machine learning analysis of real human performances, but could be extended to general, novel performance behaviors.

The remainder of this article discusses two examples of building live interactive performance systems with a mixed approach, inspired by the live algorithms framework as a guide to development. Following a bricoleur approach, many creative decisions arise in an ad hoc manner. The precise design of the systems is not therefore explained in its entirety. Instead, general methodological ideas are discussed with a view to future live algorithms practice and the nature of improvised human-computer interaction.

Two Studies Using the LAM Framework

For the research presented in this article, the creative goal is to build a real-time improvisation system with a minimal prescribed element, and to set up basic conditions for dynamical interactive behavior, leaving room for a programmer to creatively design generative musical systems around it. We can think of the system as autonomous and creative, but only insofar as its output clearly goes beyond this design process, and only a weak claim to such an achievement is made in this article.

The analysis system, *P*, remains relatively fixed, providing a suitable, minimal level of information about the acoustic environment. The internal modular system, *f*, is intended to offer a variety of rich behavioral options which can be instantly substituted for each other. The generative system, *Q*, is intended as more of a dynamical composition.

Blackwell and Young's *PfQ* framework (Blackwell and Young 2004, 2005) is applied in this article as follows. A set of audio analysis tools gather low- and high-level feature data from the incoming real-time audio and map this information to a stream of real-valued vectors, where each element in the vector corresponds to a feature. This stream is used to drive an abstract dynamical system. In the first study, the system is a continuous-time recurrent neural network. In the second study, it is a decision tree. In both cases, the system has been evolved using a genetic algorithm in an abstract artificial environment so as to exhibit certain dynamical properties.

The abstract dynamical system is used to drive a generative music system, which has been hand-coded. The system is built in the order of the information flow between these elements: The analysis module is designed first, then the behavioral module is designed to produce appropriate dynamical output in response to the analysis module, and finally, the generative module can be "composed" to produce certain musical responses. In both studies, a number of variations for the generative system were considered.

First Study

In the first study, a continuous-time recurrent neural network (CTRNN) (Beer 1995, 1996) was used as the behavioral module. The CTRNN both receives and outputs vectors of real-valued numbers in the range [–1,1] (internal neural signals within the CTRNN are also of this form).

Behavioral Module

CTRNNs provide a powerful mechanism for driving novel reactive behavior, which can be adapted to exhibit specific behavioral goals. A CTRNN is a network of simple artificial neurons, each of which processes a floating-point value and maintains a state. The network nodes are interconnected by a set of directional weighted synapses. Each node adds up the weighted values of the activations at each input and processes the result to produce an output value, which is in turn passed to other nodes via their weighted synapses. The update process is repeated synchronously for all nodes at a fast rate (on the order of once every 10 msec), to achieve a smooth continuous stream of activation behavior. Using a sigmoid function on the output of each node, the output values of nodes are naturally constrained to the range [–1,1]. CTRNNs can therefore be

understood as black-box behaviors that operate on a vector of N incoming real values in the range [–1,1] to produce a vector of M outgoing real values, also in the range [–1,1].

Researchers have had success in using artificial evolutionary optimization to discover designs for CTRNNs that achieve specific tasks (e.g., Beer 1996). Artificial evolution here simply refers to the gradual replication, with mutations in the design, of a population of CTRNNs, where the choice of which members to replicate, and which to discard, is based on success at a given task. This is determined by designing simulated environments in which candidate CTRNNs are iteratively tested and compared according to their ability to perform the given function (Beer 1996; Slocum, Downey, and Beer 2000).

Due to its complex and continuous behavior, the CTRNN is not a practical choice for musical systems that have specific behavioral problems to solve, such as harmonization or high-level melodic patterning. However, effective use of CTRNNs to achieve compelling patterns for creative musical tasks has been demonstrated by Bown and Lexer (2006). CTRNNs were evolved to achieve the simple behavioral goal that, in the presence of external activity, they would become dynamically active, while without external activity, they would come to rest. This behavior is sometimes referred to as a dynamical reservoir (Jaeger 2003). By analogy with the behavior of water, the network can be activated by an external stimulus and eventually (but not immediately) come to rest in the absence of activation. By the same analogy, the specific pattern of activation given to the network can also result in categorically different patterns of behavior.

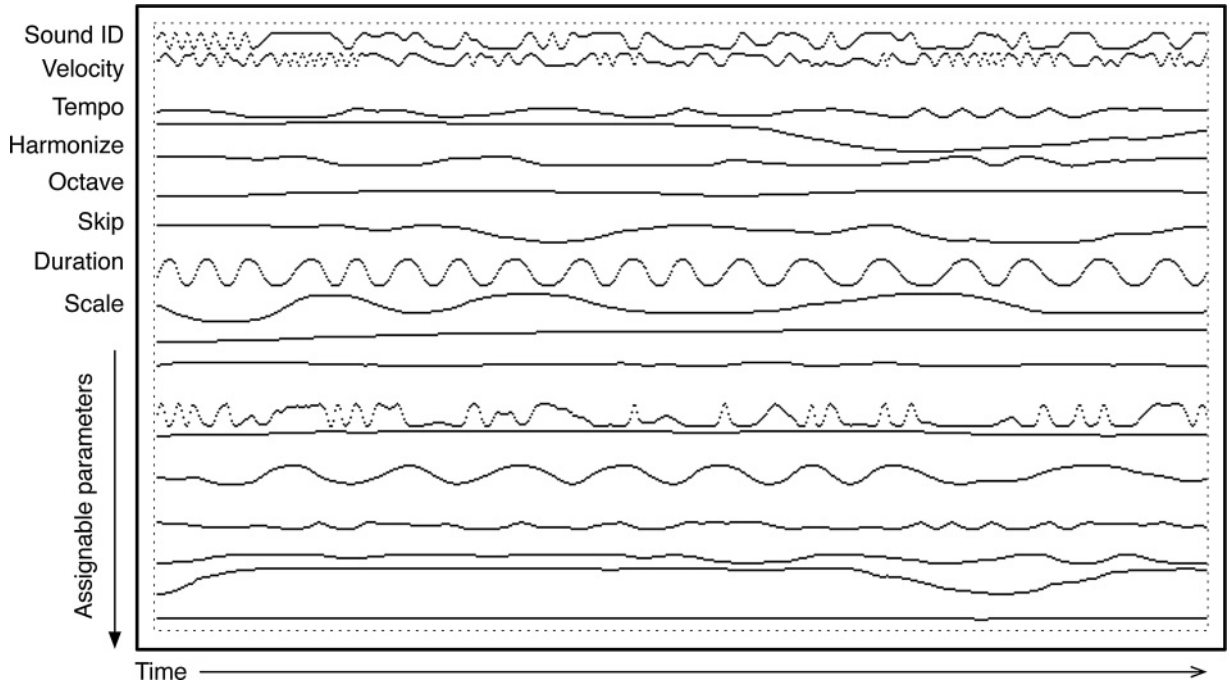
Complex CTRNNs that are effective at achieving such simple behaviors can be evolved. A genetic algorithm was applied to a population of fixed-size CTRNNs, using a fixed multi-objective fitness function. Networks were selected in the evolutionary processes for their success at acting with the responsive properties of dynamical reservoirs, and for exhibiting simple repetitive outputs in the context of repetitive inputs. Inevitably, the resulting networks exhibited additional behavioral traits not specified by the evolutionary goal. These can also be described as idiosyncratic, or having character, partly due to their random origins, partly inherent in the nature of CTRNNs, and potentially partly due to the resolution of internal contradictions within the evolved system. Such networks are musically interesting because they mediate between a desired target behavior and something beyond the intentions of the designer. Unlike simply introducing randomness into the system, these behavioral artifacts are deterministic, consistent, and ultimately learnable, giving them a characteristic flavor.

CTRNNs are also related to various forms of neural networks that have been proposed as mechanisms for automating musical entrainment (Jones and Boltz 1989; Large and Palmer 2002), such as those studied by Large and Jones (1999), Large, Fink, and Kelso (2002), and Eck (2002). Although not studied directly in terms of their capability to entrain, informal observations of CTRNNs used in musical contexts demonstrated that regular input patterns could, under some circumstances, cause oscillating CTRNNs to fall into regular cycles with the same period, suggesting the kind of meaningful musical interaction that coupling might enable: even if rhythmic entrainment is not strictly achieved, the loose coupling that facilitates a fragile quasi-synchrony has strong musical appeal.

Generative Module

The CTRNN was used to drive a generative music system, which provided both an additional level of complexity and an area in which a more specific and prescribed generative musical behavior could be defined. In previous work (Bown and Lexer 2006), CTRNNs were used to directly modify the parameters of a playing sound, such as the playback position of a scrubbed granular sample player, or parameters of an FM synthesizer. CTRNNs have an elegant drifting behavior with sudden state transitions that provide a musical dynamism appropriate to the direct control of sound (see Figure 2). In the current work, the generative system was expanded to include discrete generative behavior, combined with direct control of continuous parameters.

Figure 2. Output from a CTRNN in a musical performance. Each node in the CTRNN changes its internal activation level over time in a way that is coupled to the other nodes. Activation levels range between –1 and 1. Each node is used to control one or more parameters in the performance. Some of the mappings from nodes to generative parameters are fixed, as indicated on the y axis. Others are assigned by sound elements to modulate parameters. The CTRNN's behavior demonstrates patterns of change of different output parameters over different time scales.



Discrete control was achieved by allowing one output of the network to control the rate of a clock used to trigger musical events. In previous work discrete events had been triggered by drawing the timing from the movement of the CTRNN itself, exploiting the CTRNN's inherent rhythmic behavior. The choice of the clock in the present work reflected the desire to achieve more stable rhythmic patterning. The choice of which event to trigger (if any) at any given beat was determined by the state of additional network outputs. The network's state was examined at each beat of the clock and its output parameters were mapped to integer values which determined notes, registers, and sound selections from pre-composed lists. The CTRNN's outputs were also used to determine the event's velocity and its attack and decay durations. Sets of pitches were generated using simple harmonic transitions.

Sets of sound events were generated using aesthetic evolution, using a process similar to the aesthetic evolutionary techniques used by Dahlstedt (2006) in his MutaSynth software, and inspired by the Blind Watchmaker algorithm popularized by Dawkins (1986).

The interface between the CTRNN and the generative music system is presented in Figure 3. At the onset of an event, triggered by the clock and determined by the interaction of other parameters at that moment, a new sound event was generated on the fly with additional parameters determined by the CTRNN (pitch, velocity, attack, and decay parameters, as discussed previously), and connected to the audio output of the live algorithm. Each sound event consisted of a digital signal processing network defined by a pre-evolved genotype, and chosen using a lookup table. An amplitude envelope was constructed from the attack and delay parameters, the end of which automatically triggered the removal of the sound object from the audio chain. In order to allow the system to generate very long sounds, but without exceeding sensible levels of polyphony, all active sounds were also logged by the generative music system, which could then make executive decisions based on the current polyphony of its output, such as whether to withhold a new event for

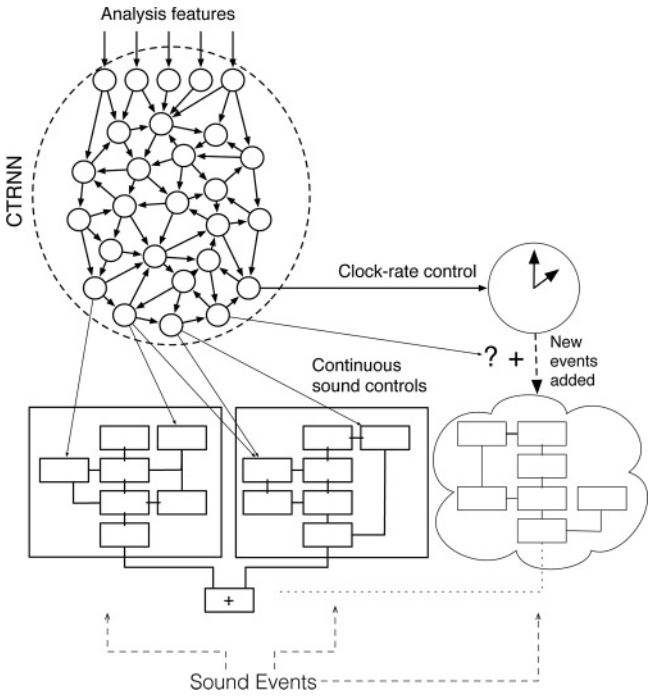


Figure 3. Diagram of relation between behavioral module (CTRNN) and generative module. Specific outputs from the network are used to determine different aspects of the generative system. One output controls the rate of the clock, which triggers sound events (the two large rectangles are existing sound events, the cloud is a sound event coming into existence). Each sound event consists of a network of digital signal processing objects (small rectangles). Another output is used to determine which event gets triggered by the clock. Additional outputs are subsequently plugged into the resulting sound event to provide real-time control of sound parameters. Parameters of different sounds which are being controlled by the same network output will move in tandem.

playing, or destroy old events in order to make way for new ones.

Performance Experience

The hybrid CTRNN live algorithm has been used at various stages of development in duets with live performers. The description herein refers only to the latest version. It was presented at a LAM symposium at Goldsmiths College in 2006 in performance with trombonist George Lewis. At a concert at London's Café OTO, following another LAM workshop at Goldsmiths in 2009, it performed in duets with flautist Finn Peters and drummer Eddie Prévost. In its latest performance, the system was used to generate a series of five short improvised performances with different sonic and behavioral characteristics for a series of concerts called Hands Free, organized in Melbourne by the author. The performances involved duets with bass clarinetist Brigid Burke and trombonist and shakuhachi player Adrian Sherriff.

The interactive experience appeared to work best when the performers played assertively over the software, with the system filling the background space and occasionally interjecting events that drew their attention and altered the trajectory of their performance. Inevitably, the system would take the lead at times. The performers were asked to focus on the overall sound as an end product, and their contribution to it, rather than directly on the interaction with the system or any expectation about what it may do. Performers reported a sense of interactivity but some unease with the nature of variation exhibited by the system. For example, although the output of the network caused the clock speed to vary, its repetitive oscillatory behavior resulted in an apparently very regular sequence of events.

Although such a complex system makes it hard to devise precise musical behaviors, the potential to manually encode the generative music system in part facilitated creative development for specific goals. A library of evolved generative sounds, samples, and evolved CTRNNs could be used to experiment rapidly with different configurations in a live context. Because these networks act holistically, changing one element influences the other elements in the system, and although this can be a cause of frustration, it can also be a driver of creative exploration. Insofar as the system is unprescribed, the potential to stumble upon new and unexpected configurations through a combination of accident, focused exploration, and problem-solving was effective.

Second Study

The system's opacity was, overall, a hindrance to the fine-tuning and development of a powerful live algorithm. The goal of the second study was to seek a system that had similar dynamical properties but was easier to understand and manipulate.

Behavioral Module

Decision trees (DTs) are generally used for classification of parametrically analyzed elements, in a similar way to neural networks (Breiman, Friedman, and Olshen 1984). Given some numerical

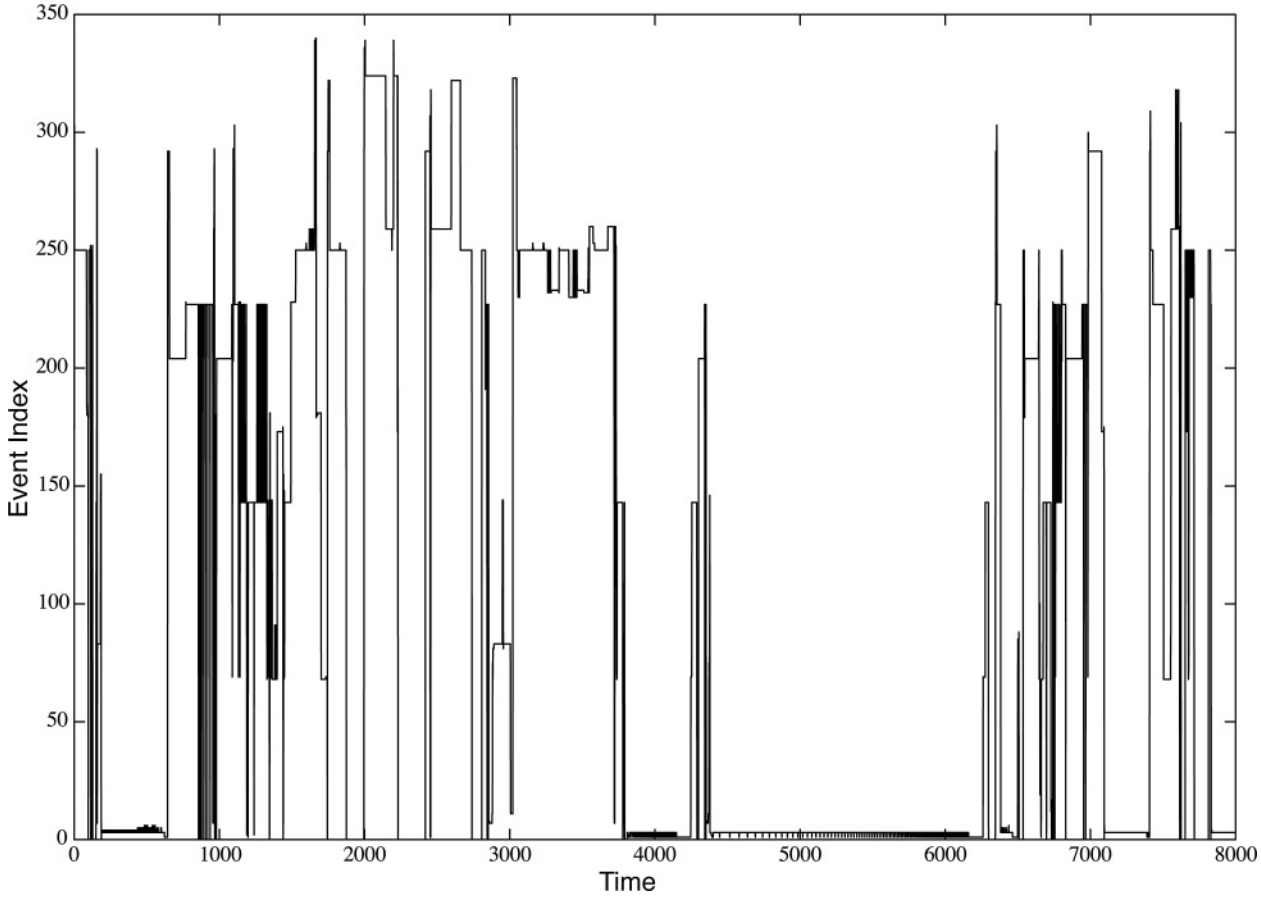
data about a class of objects, the decision tree will carve up the multidimensional space of possible data points and work out which category of this space any given data point belongs to. DTs work by iteratively asking questions of the form: "Is parameter *x* less than threshold *y*?", with each question depending on the outcome of the previous one, until a decision is reached about the state of the thing being examined. In our case, the object of analysis is real-time audio data. Optimized DTs find the shortest set of questions to reach a relevant decision, which in many cases make them more efficient than other categorization techniques. In programming terms, a DT is modeled as a data point (a real-valued vector) and a binary tree of nodes. Each node is either a branch, in which case it determines which one of two nodes is passed the data next, or a leaf, in which case it executes an action. Each branch node has an index pointing to an incoming data parameter, and a threshold against which to test that parameter. The Boolean outcome of this comparison leads to one of two follow-up nodes. At each time step, the data are updated from the audio analysis source and fed through the tree from its root.

The author is not aware of any previous use of DTs as generative, responsive dynamical systems (although they are commonly used in music information retrieval tasks). In this case, the input to the DT was a set of real-time audio features updated at a rapid rate (on the order of 10 msec), just as in the case of the CTRNN. In order to mimic the dynamical properties of the CTRNN, a hidden internal state, consisting of a set of floating-point values, was also assigned to the DT, which the DT both analyzed (in addition to the audio data) and could modify through a sequence of simple mathematical operations that was dependent on the decision made by the DT at each time step. This constantly updating internal state meant that, as with CTRNNs, DTs could be left to run without input activity and still produce dynamical patterned behavior. Ultimately, as with all bounded, discrete-state, closed dynamical systems, such a pattern of activity will end up in a periodic or fixed attractor following an initial transient sequence. As with the CTRNN, the DT's internal state is treated, in the first instance, as abstract raw material, which can then be interpreted by the generative module to different musical ends.

DTs were considered here as an appealing alternative to CTRNNs for five reasons:

1. At each time step, DTs produce a distinct discrete output—the decision. While CTRNN output can be discretized, the CTRNN moves smoothly through its output state, meaning that consecutive discrete output states will be adjacent. DTs, on the other hand, can flip from one output decision to any other as the internal state drifts into different zones. This makes for more interesting pattern generation of discrete events such as musical notes. The internal state, which is capable of gradual movement, also offers an option for continuous output, so the DT can be used to provide both continuous and discrete control.
2. It is much easier to break down and analyze the behavior of a decision tree. Whereas CTRNNs are famously cryptic, the behavior of each individual node in a DT is perfectly clear (although the emergent behavior of the overall system may remain opaque). Through brute-force state-space analysis, DTs can be pruned and optimized, and potentially merged and subdivided (although this wasn't explored in the current study).
3. Performing only a small sequence of conditional operations at each time step, DTs are very efficient, as compared with CTRNNs, which might potentially perform tens or hundreds of floating-point operations.
4. DTs can be easily evolved in such a way as to grow, from small and simple, to large and complex structures by occasionally converting leaf nodes into branch nodes. The equivalent growth mechanisms for evolved CTRNNs are more convoluted.
5. DTs can be programmed with an adaptive self-calibration behavior such that nodes that always produce the same outcome gradually shift their thresholds until they find an active decision boundary. They can

Figure 4. Output from a decision tree running with constant inputs (i.e., not listening to any musical activity). The output is an integer that indicates the decision made by the decision tree at each time step (i.e., the ID of the leaf node reached through the decision process). The output demonstrates simple rhythmic patterning and variation over time.



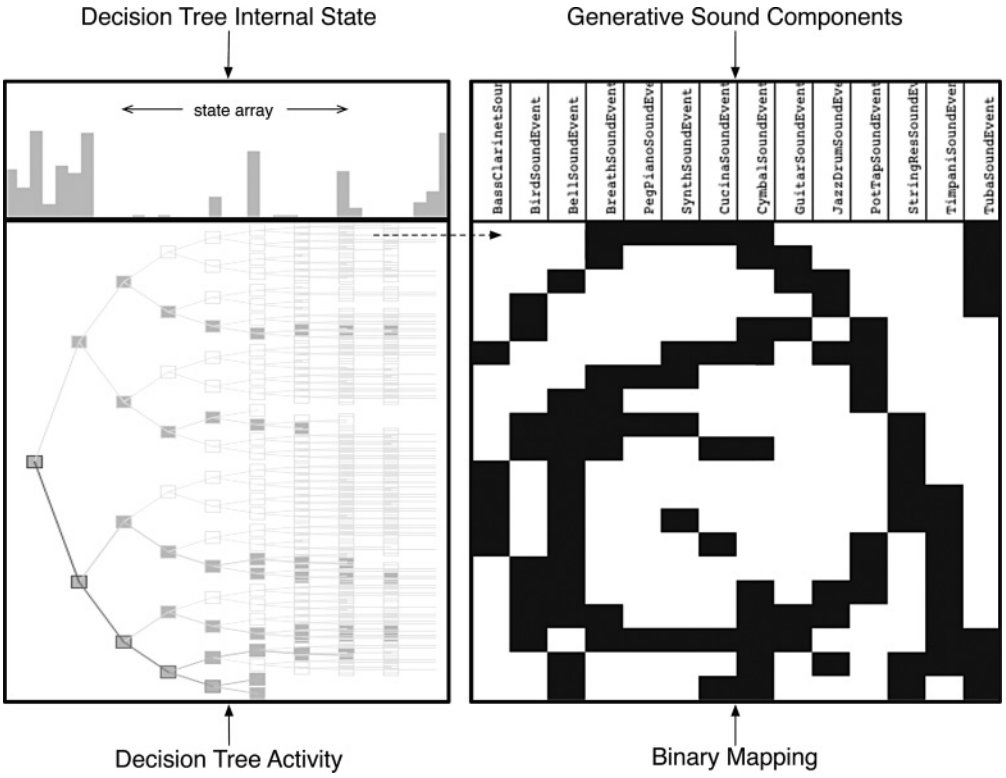
even grow adaptively in real-time in an unsupervised manner.

As with CTRNNs, DTs were evolved with abstract behavioral goals in mind. A fitness function was devised that combined a set of dynamical properties of the resulting DTs, taking into account the growth over evolutionary time of DTs due to mutation operators that occasionally added new nodes. A key dynamical property was the number of leaf nodes visited in total, which was set to be maximized in the fitness function. This target encouraged node thresholds to adapt to critical points, and the internal state update processes to keep the internal state in constant change. This established DTs that were active on their own, independent of the input that was being fed to them. However, the evolutionary simulations were run using real analyzed audio as input, and the resulting DTs were in fact very responsive to real audio inputs. An example of the decision output of an evolved DT running with no inputs is shown in Figure 4.

Generative Module

The advantages of DTs over CTRNNs listed earlier made it much easier to visualize the responsive behavior of the live algorithm to musical input and to design generative modules in a creative compositional manner. A series of generative sound event objects were created, each based around a simple musical theme. Unlike the CTRNN, the DT outputs a discrete integer as well as an array of continuous state values. This integer can be

Figure 5. Graphical interface from the decision tree live algorithm. Top left: current state the decision tree's internal state array. Bottom left: tree diagram showing recent decision activity of the decision tree (highlighted lines). A decision is made by passing from the root (left) to a leaf node (right). Top right: set of available sound events. Bottom right: grid mapping decision to selected sound event. The decision made by the decision tree (the resulting leaf node index) activates a row in the grid. For each column in the grid that is colored black at that row, the corresponding sound event is triggered, with additional parameters determined by the exact index of the decision node and the current internal state of the decision tree. By mixing multiple sound events in one row a complex instrument can be built with elements that vary in parallel. A user can read a stored decision tree file and choose a graphical mapping in real time, or load an entire scene saved from memory.



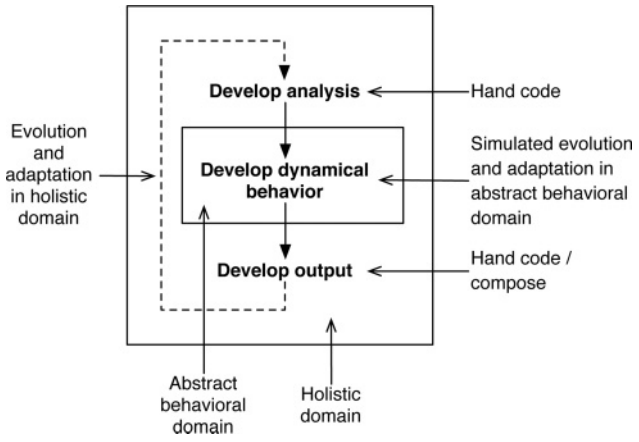
used within the generative objects for more discrete control of output (e.g., to select a sample or a pitch value). In one example, the generative sound event object played back birdsong using granular playback, with the DT controlling the parameters of the granulation. Another played timpani from a bank of samples with additional modulation of filter parameters, pitch, and velocity. Hand-coded filtering of the incoming integers was used to dictate a general rhythmic behavior for each of these elements where necessary; this was done such that variation in the behavior of the DT still caused rhythmic variation.

The event objects were then combined in an interactive grid such that any DT decision could trigger any set of event objects (see Figure 5). At each timestep, each event object being triggered was passed the current decision ID, plus the current internal state of the DT. Event objects generally performed some throttling on the input data, which came at a control rate, and typically only activated when a change was detected in the decision ID. Decision IDs could be used to choose discrete events, and the internal state could be used to modulate continuous data, providing a rich array of performance control options.

Performance Experience

The DT-based live algorithm was premiered at the 2010 North Sea Jazz Festival, as part of an improvised performance involving the trumpeter Tom Arthurs, the bass clarinetist Lothar Ohlmeier, and other live algorithm software by Sam Britton (not running at the same time as this software). It was also shown at the 2010 Australasian Computer Music Conference, with bass clarinetist Brigid Burke. In both cases the software was run without intervention from a human controller, except to occasionally change

Figure 6. Workflow devised from LAM framework. Following the signal path through the system from the top, a set of analysis tools is first developed using standard libraries and manual calibration (top right). Then the dynamical behavior is developed using simulated musical data, and an evolutionary or learning procedure (middle right). The target behavior of the system is defined in terms of abstract dynamical properties. Then the generative module is developed manually, as a compositional process that uses the abstract dynamical behavior produced by the system (bottom right). Ultimately, evolution and adaptation can also occur in the holistic domain (left) as well as in the abstract behavioral domain.



"scenes"; a scene consists of a DT behavior and a configuration of mappings from DT outputs to sounds (as in the grid in the bottom right of Figure 5). The musicians who had worked with both systems tended to find the DT-based system to be more responsive than the CTRNN-based systems. Collaborative work with musicians to develop appropriate interactive compositions was also far easier from a programming point of view, due to the more comprehensible behavior of the DTs. Although DTs had not been directly evolved to respond to changing audio input, the responsive nature of the DT was more evident than with the CTRNN. This was partly because a sudden change in input values tended to trigger a change in the DT, as one node in the tree flips state due to a crossed threshold; DTs find fixed or cyclic attractors in the context of a stable input state, but will be immediately perturbed from these attractors by a change in that input state. Although CTRNNs can also exhibit this behavior, it was harder to achieve in practice. This immediate interactive sense is a simplistic version of what has been described by Young and Bown (2010) as *shadowing*, and can be seen as a basic but effective form of event following. The author found this to be a convenient way to offer the performer a sense of interactive effect, but without directly programming "triggering" behaviors into the system, relying instead on the dynamical properties of a complex, evolved behavioral module, which may have a more nuanced responsive behavior.

Conclusions and Proposals

The studies presented in this article explore the efficacy of the live algorithms approach to developing creative and autonomous musical software systems, combining both technical and compositional challenges. The results suggest appropriate tools and a working methodology for developing an ongoing series of live algorithms works. The notion of developing such systems simultaneously at two distinct levels of modularity was explored. At one level, complete live algorithms interact with their environments via digitized musical signals, offering the potential to be placed in musical performance contexts. At the other, lower level, behavioral subsystems (f) can be explored in terms of their abstract dynamical properties, learning or evolution can be used to control the precise behavior of these subsystems in abstract terms, and appropriate interfaces can be worked out between these subsystems and the enclosing elements (P and Q) that, with them, make up complete live algorithms.

The work presented here suggests that this methodology is sound and fruitful, and offers ways in which different design methods can be combined to produce complete live algorithms. It proposes an approach to live algorithm design that follows the signal path from a hard-wired analysis system, P, through an abstract, evolved behavioral module, f, to a creatively composed generative music system, Q (see Figure 6). The behavioral variation evidenced in these studies suggests that families of live algorithm systems can then be explored in a productive modular manner.

The differences between the behavior and practical usability of CTRNNs and DTs illustrate a number of subtle issues concerning designing behavior into live algorithms. For a number of reasons, DTs were the more usable type of system, particularly because they produced discrete patterned output, and partly because their behavior could be deconstructed and understood slightly more easily than that of CTRNNs. However, in both cases an elusive goal is to discover successful approaches to creating behaviors through evolution or learning or some other training process, which can be practically used and modified in a creative context. For

this reason, at present neither system is considered better, because each possesses musical qualities that may be of interest to different people in different contexts. Instead, an eclectic approach to designing live algorithms of many different varieties is sought. Although the present work does not provide concrete answers to this problem, it hopefully takes steps towards understanding how such a modular approach might ultimately achieve this goal.

Further information, examples of the software presented in this article, and musical output can be found at www.olliebown.com/cmj_lam as well as on the forthcoming Winter 2011 *Computer Music Journal* DVD.

Acknowledgments

This article has benefitted greatly from the comments of the two anonymous reviewers, and from fruitful discussions with Tim Blackwell, Sam Britton, Michael Young, the members of CEMA, and the attendees of the 2009 LAM workshop. I also thank Juha van't Zehe for organizing a significant public presentation of these systems at the North Sea Jazz Festival, as well as the various performers named within who have contributed to this project. This research was supported by an Australian Research Council Discovery Project Grant, DP0877320.

References

Assayag, G., G. Block, and M. Chemillier. 2006. "OMAX-OFON." In *Proceedings of Sound and Music Computing (SMC) 2006*. Available on-line at smcnetwork.org/node/1090. Accessed April 2011.

Beer, R. D. 1995. "On the Dynamics of Small Continuous Recurrent Neural Networks." *Adaptive Behavior* 3(4):469–509.

Beer, R. D. 1996. "Toward the Evolution of Dynamical Neural Networks for Minimally Cognitive Behavior." In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*. Cambridge, Massachusettes: MIT Press, pp. 421–429.

Biles, J. A. 1994. "GenJam: A Genetic Algorithm for Generating Jazz Solos." In *Proceedings of the International Computer Music Conference*, pp. 131–137.

Biles, J. A. 2001. "Autonomous GenJam: Eliminating the Fitness Bottleneck by Eliminating Fitness." Available on-line at www.ist.rit.edu/~jab/GenJam.html.

Blackwell, T., and M. Young. 2004. "Self-Organised Music." *Organised Sound* 9(2):137–150.

Blackwell, T., and M. Young. 2005. "Live Algorithms." Available on-line at www.timblackwell.com.

Bown, O., A. Eldridge, and J. McCormack. 2009. "Understanding Interaction in Contemporary Digital Music: From Instruments to Behavioural Objects." *Organised Sound* 14(2):188–196.

Bown, O. and S. Lexer. 2006. "Continuous-Time Recurrent Neural Networks for Generative and Interactive Musical Performance." In F. Rothlauf and J. Branke, eds. *Applications of Evolutionary Computing, EvoWorkshops 2006 Proceedings*, pp. 652–663.

Breiman, L., J. H. Friedman, and R. A. Olshen. 1984. *Classification and Regression Trees*. Pacific Grove, California: Wadsworth and Brooks.

Cope, D. 1992. "Computer Modeling of Musical Intelligence in EMI." *Computer Music Journal* 16(2):69–83.

Dahlstedt, P. 2006. "A Mutasynth in Parameter Space: Interactive Composition through Evolution." *Organised Sound* 6(2):121–124.

Dawkins, R. 1986. *The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design*. London: Penguin.

Di Scipio, A. 2003. "Sound is the Interface: From Interactive to Ecosystemic Signal Processing." *Organised Sound* 8(3):269–277.

Eck, D. S. 2002. "Real-Time Musical Beat Induction with Spiking Neural Networks." Technical Report No. IDSIA-22-02. Manno, Switzerland: Instituto Dalle Molle di studi sull' intelligenza artificiale. Available on-line at citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.3633.

Jaeger, H. 2003. "Adaptive Nonlinear System Identification with Echo State Networks." *Advances in Neural Information Processing Systems* 15:593–600.

Jones, M. R., and M. Boltz. 1989. "Dynamic Attending and Responses to Time." *Psychological Review* 96(3):459–491.

Large, E. W., and M. R. Jones. 1999. "The Dynamics of Attending: How People Track Time-Varying Events." *Psychological Review* 10(1):119–159.

Large, E. W., and C. Palmer. 2002. "Perceiving Temporal Regularity in Music." *Cognitive Science* 26(1):1–37.

Large, E. W., P. Fink, and J. A. S. Kelso. 2002. "Tracking Simple and Complex Sequences." *Psychological Research* 66(1):3–17.

Lewis, G. E. 2000. "Too Many Notes: Computers, Complexity and Culture in Voyager." *Leonardo Music Journal* 10:33–39.
McLean, A., and Wiggins, G. A. 2010. "Bricolage Programming in the Creative Arts." Available on-line at http://yaxu.org/writing/ppig.pdf. Accessed 9 June 2011.
Rowe, R. 1993. *Interactive Music Systems*. Cambridge, Massachusettes: MIT Press.
Rowe, R. 2001. *Machine Musicianship*. Cambridge, Massachusetts: MIT Press.
Slocum, A. C., D.C. Downey, and R. D. Beer. 2000. "Further Experiments in the Evolution of Minimally Cognitive Behavior: From Perceiving Affordances to Selective Attention." In J. Meyer, et al., eds. *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*. Cambridge, Massachusettes: MIT Press, pp. 430–439.
Winkler, Todd. 1988. *Composing Interactive Music*. Cambridge, Massachusetts: MIT Press.
Young, M., and O. Bown. 2010. "Clap-Along: A Negotiation Strategy for Creative Musical Interaction with Computational Systems." In *Proceedings of the First International Conference on Computational Creativity*, pp. 215–222.