LOLbot: Machine Musicianship in Laptop Ensembles

Sidharth Subramanian
Georgia Tech Center for Music Technology
840 McMillan Street
Atlanta GA 30332-0456
ssubramanian30@gatech.edu

Jason Freeman
Georgia Tech Center for Music Technology
840 McMillan Street
Atlanta GA 30332-0456
jason.freeman@gatech.edu

Scott McCoid
Georgia Tech Center for Music Technology
840 McMillan Street
Atlanta GA 30332-0456
smccoid@gatech.edu

ABSTRACT
This paper describes a recent addition to LOLC, a text-based environment for collaborative improvisation for laptop ensembles, incorporating a machine musician that plays along with human performers. The machine musician LOLbot analyses the patterns created by human performers and the composite music they create as they are layered in performance. Based on user specified settings, LOLbot chooses appropriate patterns to play with the ensemble, either to add contrast to the existing performance or to be coherent with the rhythmic structure of the performance. The paper describes the background and motivations of the project, outlines the design of the original LOLC environment and describes the architecture and implementation of LOLbot.

Keywords
Machine Musicianship, Live Coding, Laptop Orchestra

1. INTRODUCTION
This paper describes recent additions to LOLC [5], a text-based environment for collaborative improvisation for laptop ensembles, which integrate machine musicians into the LOLC environment. LOLC's text commands and networked development environment enable the creation, transformation, and sharing of short musical motives built from one-shot pre-recorded audio files. LOLbot is an environment that interfaces with the existing LOLC client software, creating a machine musician that mimics some of the activities of a human LOLC laptop musician: it analyzes the performance of the laptop ensemble and uses that analysis to determine which LOLC commands to type to generate its own music.

Our goals in adding a machine musician into LOLC were threefold: (1) model how human performers improvise using LOLC, especially in terms of how they analyze rhythms and use this information in the improvisation process; (2) add a new dynamic to the performance of LOLC, given the highly computational approach of the machine musician as compared to that of the human performers; and (3) provide a practice tool for musicians who want to learn LOLC or improve their improvisational skills in the context of LOLC.

This paper describes the background and context in which LOLC and LOLbot were developed; briefly overviews the LOLC environment; describes the design and implementation of LOLbot; and discusses future possibilities.

NIME'12, May 21-23, 2012, University of Michigan, Ann Arbor.

2. BACKGROUND
Robert Rowe defines machine musicianship as the process of "analyzing, performing and composing music with computers."[10]. Music is generally associated with the human creative process, and the advent of using the processing capabilities of a machine can be thought of as another avenue to inspire creativity in performance. Some examples of systems that analyze and react to live performers include Lewis's *Voyager* [7], Pachet's *Continuator* [9] and Weinberg's *Shimon* [6].

The analysis of human performance has always been a significant aspect of machine musicianship, using techniques from music information retrieval, pattern recognition, and machine learning. Since LOLC is a text-based environment, and since all LOLC musical patterns are encoded symbolically, the analysis of the music can be done in the same realm in which it was created — the syntax of LOLC. The machine musician has direct access to the symbolic musical motives created by the performers and consequently, the entire performance. Early examples of using pattern matching and text analysis in an effective way to simulate human intelligence go back to Eliza [12]. As described in [3], other examples include Instant Messaging bots that are found in chat systems like IRC where bots take part in conversation over instant messaging; and instant messaging bots in programming environments that describe program states and changes, allowing developers to collaborate while programming and debugging. Unlike these examples, the musical motives created by LOLC performers are in a syntax that is specific to LOLC and thus the system's standard parser can be used in place of any natural language recognition algorithms.

In the realm of computer music, members of the live coding community have recently begun to explore the algorithmic generation of live code; ixi lang's Autocode feature serves as a prominent example [8]. While LOLC is not itself a live coding language, we were nevertheless influenced by approaches such as that in ixi lang as we designed LOLbot.

3. LOLC
Inspired by computer music languages such as Impromptu [1] and live coding systems such as jitlib for SuperCollider [4] and the Co-Audicle for ChucK [11], LOLC [5] was designed to be a text-based performance environment (though not a Turing-complete programming language) that encourages musicians in a laptop ensemble to improvise through the composition and sharing of rhythmic motives. Influenced by works by early network music groups like the Hub's [2] *Borrowing and Stealing*, LOLC encourages performers to share their code and the patterns they create, giving the ensemble access to a growing collection of patterns to use, transform and loop as they improvise. (LOLC was presented as a performance at NIME 2011).

The server component of LOLC is responsible for the synchronization aspects of the ensemble's performance. The

server maintains time synchronization through a shared clock; a pattern library consisting of all the patterns created thus far in performance, their definitions and their scheduling information; and an instant-messaging-style communication channel through which the ensemble members collaborate. The client component (see Figure 1) presents an instant-messaging-style interface that shows both commands and chat messages from everyone in the ensemble. Performers can use this interface for the creation and scheduling of patterns and for access to the library of patterns created thus far in a performance. LOLC also includes a visualization (see Figure 2) for audience members to watch; the visualization shows the performers' activities in terms of playing patterns, sharing/borrowing patterns from others and communication between performers.

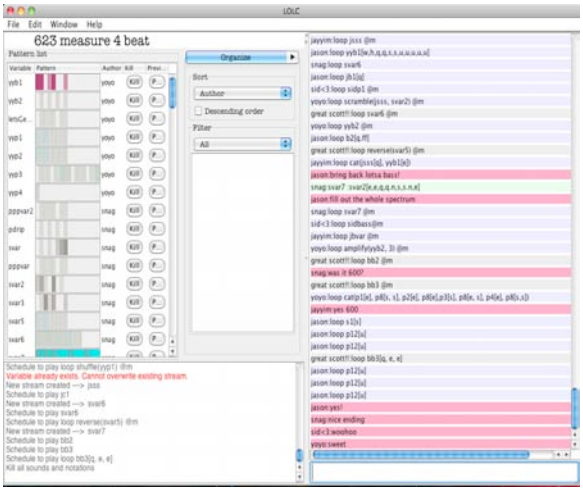

Figure 1. LOLC Client Component

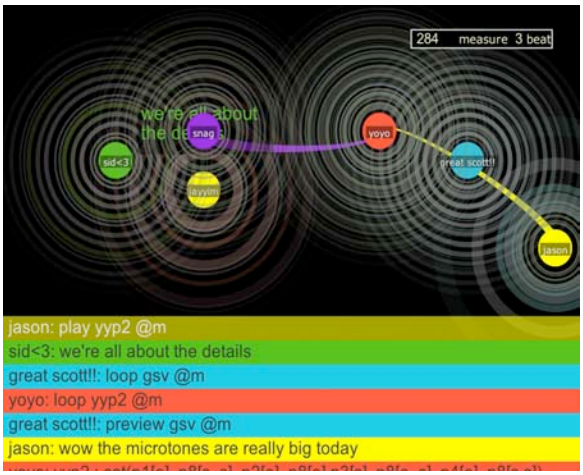

Figure 2: LOLC Server Visualization

Pre-recorded one-shot sound samples varying in timbre and pitch are available to the performers to choose from. Performers can syntactically create rhythmic patterns using these sound samples. Once the patterns have been created, they can be scheduled to be played at any point during the performance, with the option of looping them as many times as desired. Shown below is an example of the pattern creation process:

```
mySound: "a1.wav"
myPattern: mySound[q.ff,e.ff,s.n,h]
```

This example shows the assignment of a user-specified WAV file to a variable, and the creation of a pattern using this WAV file. The pattern 'myPattern' consists of a quarter and eighth note at fortissimo, a sixteenth note rest and half note which is at the default mezzo-forte (since the dynamic is not specified). Patterns can be defined to be of any length from 128th notes to arbitrarily long durations. Once a pattern is created, it can be scheduled to play:

```
play myPattern @nextMeasure
```

The example above shows how a pattern can be scheduled to play once at the next measure. LOLC allows for great flexibility in the scheduling of patterns, allowing performers to loop patterns at particular measures, controlling how many times the pattern loops:

```
loop myPattern @512 ~14
```

The example above would schedule the pattern 'myPattern' to be played at measure 512 a total number of 14 times. This need not correspond to 14 measures, in cases of patterns like 'myPattern' that are less than 1 measure long.

As patterns are created, they are stored in the pattern library that is accessible to all performers, who can then build upon these patterns through transformations, mimicking the collaboration of improvising acoustic musicians. An example of one of thirteen supported transformations is shown below:

```
myPatternTransf: reverse(myPattern)
```

The example above creates the pattern 'myPatternTransf' that is the reverse of the order of the sequence of 'myPattern'.

4. MACHINE MUSICIANSHIP IN LOLC

The design of LOLbot is derived from the analytical as well as the improvisational process of human performers. LOLbot is designed to model how human performers perceive the music generated by the rest of the ensemble in terms of accents, and to improvise using this information with normal LOLC syntax. LOLbot does this by analyzing the music generated by the rest of the ensemble in detail, and playing patterns created by the ensemble, chosen based on its input parameters. LOLbot even exists on its own laptop using its own version of the LOLC client software.

Despite being modeled on human performers, LOLbot differs from human performers in the nature of its analysis and compositional process. LOLbot uses its memory to store detailed information on every pattern played during the performance, so that each pattern is considered every time it plays. LOLbot uses its machine precision and computational power to analyze the performance at the 16^{th} note level and uses pattern matching algorithms to find the most suitable pattern among all patterns played by the entire ensemble. This creates a new performance dynamic in LOLC – contrasting the approaches of LOLbot and a human performer to the processes of analysis and improvisation. While LOLbot adopts an approach that is based purely on computation, human performers' approaches are based on various factors like their perception, cognition and their own musical aesthetics.

LOLbot also lends itself to be a practice tool for performers who wish to learn how to improvise better. Performers can select whether they want LOLbot to reinforce the overall rhythmic nature of the performance or add contrast to the performance using the Coherence/Contrast slider. They can then create and schedule various patterns and observe how the patterns they created are used by LOLbot to either add contrast to the existing performance or reinforce it. By comparing the output of LOLbot to their own perception of the performance, they can discover new approaches to improvisatory collaboration with LOLC.

LOLbot's approach to machine musicianship focuses on rhythm and borrowing. The syntax of LOLC lends itself to

the creation of rhythmic patterns, which when layered upon each other also form new rhythms. Performers usually create patterns with a specific rhythmic intent and hence an analysis of the rhythms would be a natural avenue for a machine musician. LOLC is inspired by ensemble-based collaboration, where it is very common for human performers to "borrow" patterns created by other players and either use these patterns verbatim or apply transformations to them. This means that through the course of a performance, a central pool of patterns is created that is available to all performers and the synergy associated with this level of collaboration is one of the most interesting aspects of LOLC. So borrowing is the most natural function that would be associated with a new performer, who is exploring the possibilities LOLC has to offer. The design of LOLbot reflects this by re-using patterns created by performers.

5. TECHNICAL IMPLEMENTATION

Built upon the existing architecture of LOLC, LOLbot is implemented using Java. The implementation of LOLbot consists of 3 main components: Pattern Storage, Pattern and Performance Analysis and Pattern Matching. LOLbot has a simple user interface that allows users to set its input parameters.

5.1 Pattern Storage

There are two forms of storage implemented in LOLbot. Every pattern is first analyzed and represented in terms of its accent structure (1's and 0's). The patterns are then stored in a hash table, with the pattern accent structure as the key, storing all the patterns with the same accent structure under one key.

LOLbot also maintains a time-based, list-like data structure — the 'view' — that stores the amplitude of all the sounds played at any sixteenth note of the performance. This is so that at any point LOLbot has the capability to ascertain the accent structure at some measure in the future, based on all the patterns that are scheduled at that point. The 'view' is updated as the musicians schedule more patterns.

5.2 Pattern and Performance Analysis

In most performances, musicians using LOLC create motives that are highly rhythmic in nature (see section 3).

A sequence of 1's and 0's is used as the representation system for any pattern, if an accent is defined as any part of a pattern that has amplitude (based on the dynamics specified in LOLC) equal to or greater than the average amplitude of the pattern, and then a '1' in the sequence indicates an accent. A sixteenth-note is considered, as the basic unit for analysis; an eighth note is represented by '10', a quarter note by '1000' and so on. A pattern comprised of 4 quarter notes of the same dynamic level would be represented at the sixteenth note level as 1000100010001000. For patterns consisting of sounds with durations smaller than 16^{th} notes, the amplitudes of those sounds are added to the enclosing 16^{th} note. Hence, all the analysis is done at the 16^{th} note level.

5.3 Pattern Matching

LOLbot has access to all the patterns created by human performers in a performance, and it is from this database of patterns that LOLbot must choose an appropriate pattern to play.In order to choose a pattern to play, the metric of coherence/contrast is used. This metric can be set and changed using a slider on the LOLC client interface. The coherence/contrast slider has a range of values from 0 to 1. The value of 1 represents maximum coherence and the value of 0 represents maximum contrast. A pattern is said to be coherent with the performance if its accents match the 'perceived' accents at the measure it is scheduled at. The metric of contrast is the opposite of coherence and represents how little the accents match.

LOLbot acts once every 16 measures so that the frequency of its actions loosely mirrors the density of activities of human performers. Hence, every 16 measures, LOLbot checks the 'view' (described in 5.1) which stores the sum of the amplitudes of all the sounds being played at each 16th note, for the next 2 measures. LOLbot then converts these 2 measures into the accent representation system (described in 5.2) to get the measure accent structure. (Through multiple performances of LOLC, we found that users often created patterns that were between 1 and 2 measures long, hence a 2 measure long analysis window was adopted).

As described in the section 5.1, all the patterns created by human performers are stored in a hash table using the accent structures as the keys. So the calculated measure accent structure is then compared against the stored pattern accent structures in the hash table. We define a *contextual hamming distance* (CHD) metric to order the stored patterns. The *contextual hamming distance* is defined as the number of bits for which the measure accent structure is 1(accent) and the pattern accent structure is 0(no accent) i.e. the number of accents in the measure accent structure that are not matched by the pattern. All the stored pattern accent structures are ordered by their CHD, and in cases that 2 of them have the same CHD, the hamming distance between the pattern's accent structure and the measure accent structure is used to order them.

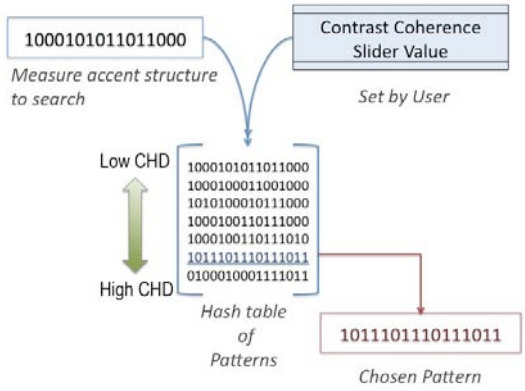

Figure 3: Choosing Patterns

Figure 3 shows how the process of pattern matching takes place. Depending on the measure accent structure being searched for, and the value of contrast/coherence that is set by the user, one of the stored pattern accent structures in the hash table is chosen. If the coherence/contrast slider is set to coherence, LOLbot chooses the pattern accent structure that corresponds to a low CHD from the measure accent structure and is hence the one that most closely matches the accents. Alternatively, if the coherence/contrast slider is set to high contrast, LOLbot chooses the pattern accent structure that corresponds to a high CHD. The slider can take intermediate values in between 0(contrast) and 1(coherence). Since all the stored pattern accent structures are ordered in terms of their CHD, depending on the number of different accent structures, each one would correspond to a certain proportional range on the slider and be chosen if the slider is set within that range. In figure 3, a pattern with a higher CHD is chosen, indicating high contrast and hence a low value on the coherence/contrast slider.

If there is more than 1 pattern with the same accent structure, a nearest neighbor algorithm is used to calculate the

'distance' between the amplitudes of the pattern and the cumulative amplitude of the 'view', at the sixteenth note level. The chosen pattern is scheduled to be played at the measure in question by sending it to the interpreter of LOLC, e.g. if the pattern chosen to be played is 'myPattern3' and the measure in question was 64, LOLbot sends the following command to the LOLC interpreter:

```
play myPattern3 @ 64
```

5.4 User Interface and Operation

The user interface of LOLbot (see Figure 4) consists of: (1) an 'enable' switch to turn LOLbot on/off, to give users the option to perform with or without LOLbot at their discretion; and (2) a contrast/coherence slider that gives the user the option to control what kind of patterns LOLbot chooses and can be modified during the performance. During performance, LOLbot runs on a separate laptop, running a dedicated copy of LOLC, and is displayed in other players' interfaces and in the video projection just as any other LOLC musician.

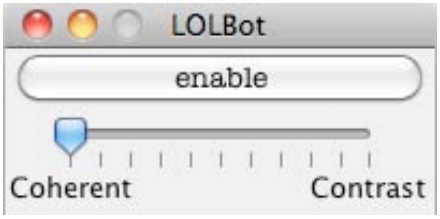

Figure 4: User Interface of LOLbot

6. CONCLUSION AND FUTURE WORK

Currently, LOLbot has been implemented using pattern matching techniques and analyses LOLC performances in terms of accents and rhythm. This is largely inspired by the inherent rhythmic nature of performance in LOLC and the syntax of LOLC that lends itself to this particular kind of rhythmic performance. This analysis of rhythm and accents is the first step towards our objective of modeling how human performers improvise using LOLC. We aim to model how performers use more advanced techniques like transformations in the future.

We tested LOLbot by means of practice performances - where 3 performers improvised using LOLC as they would in a real performance, creating and borrowing patterns. LOLbot was run on a dedicated computer as the fourth performer. Through multiple practice performances using LOLbot, we were able to gain insight into the rhythmic structure of the performance at different points of time, by modifying the coherence/contrast slider value and seeing the effect on the patterns chosen by LOLbot. This in particular allowed us to see which patterns could contribute to the performance at different points of time and hence use transformations of those patterns to create interesting rhythms. Through these practice performances, we found some limitations of LOLbot – patterns that fit more generic accent structures were chosen more often than others since they were similar to the overall rhythmic structure at many points in a performance. This could be reduced by use of the coherence/contrast slider but we aim to address this concern in the future by adjusting the distance metric based on the number of times a pattern has previously been chosen by LOLbot.

Future work for LOLbot includes applying transformations on patterns from the pattern library to have a larger selection of patterns to choose from. The process of applying transformations to create new patterns is the next step in the emulation of human performance in LOLC. This serves as an ideal precursor to multiple machine musicians in a single laptop ensemble.

We hope to use LOLbot in performances and evaluate how the use of a machine musician changes the course of the performance, and how human performers react differently in a performance with LOLbot.

7. ACKNOWLEDGMENTS

LOLC is supported by a grant from the National Science Foundation as part of a larger research project on musical improvisation in performance and education (NSF CreativeIT 0855758). We thank Akito Van Troyer, Andrew Colella, Jung-Bin Jay Yim, Sang Won Lee and Shannon Yao for their important contributions to the development of LOLC. Additional information is available at http://www.jasonfreeman.net/lolc/

8. REFERENCES

[1] Brown, A.R. and Sorenson, A.C. Interacting with generative music through Live Coding. *Contemporary Music Review*, 2009. 28(1). pp. 17-29.

[2] Brown, C. and Bischoff, J. Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area. 2002. Available online at http://crossfade.walkerart.org /brownbischoff/IndigenoustotheNetPrint.html.

[3] Chan, S., Hill, B. and Yardi, S. Instant Messaging Bots: Accountability and Peripheral Participation for textual user interfaces. *In Proceedings of the 2005 International ACM SIGGROUP Conference on supporting group work* (Sanibel Island, Florida, USA, November 06-09, 2005), GROUP '05. ACM Press, New York, NY, pages 113-115.

[4] Collins, N.,McLean, A., Rohrhuber, J. and Ward, A. Live coding in laptop performance. *Organised Sound*, December 2003. 8(3). 321-330.

[5] Freeman, J. and Van Troyer, A. Collaborative textual improvisation in a laptop ensemble. Computer Music Journal, 2011. 35(2). 8-21.

[6] Hoffman, G. and Weinberg, G. Shimon: An Interactive Improvisational Robotic Marimba Player in *Extended Abstracts Proceedings of International ACM Computer Human Interaction Conference (CHI 10)*, Atlanta, Georgia.

[7] Lewis, G.E. Too Many Notes: Computers, Complexity and Culture in Voyager*, Leonardo Music Journal, vol.10*, pp. 33-39, 2000.

[8] Magnusson, T. Designing Constraints: Composing and Performing with Digital Music Systems, Computer Music Journal, 2010, 34(4) , 62-73.

[9] Pachet, F. The Continuator: Musical Interaction with Style *In Proceedings of the 2002 International Computer Music Conference*, Goteborg, Sweden, 2002.

[10] Rowe, R., *Machine Musicianship*, Cambridge, MA: MIT Press, 2004.

[11] Wang, G., Misra, A., Davidson, P., and Cook, P.R. CoAudicle: A Collaborative Audio Programming Space. in *Proceedings of the International Computer Music Conference*, Barcelona, Spain, 2005, 331– 334.

[12] Weizenbaum, J. ELIZA – A Computer Program for the Study of Natural Language Communication between Man and Machine, *Communications of the ACM, Volume 9, Number 1* (January 1966): 36-45.