Automatic Rhythmic Performance in Max/MSP: the kin.rhythmicator

George Sioros
University of Porto (Faculty of Engineering)
and INESC - Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
gsioros@gmail.com

Carlos Guedes
University of Porto (Faculty of Engineering)
and INESC – Porto
Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal
cguedes@fe.up.pt

ABSTRACT

We introduce a novel algorithm for automatically generating rhythms in real time in a certain meter. The generated rhythms are "generic" in the sense that they are characteristic of each time signature without belonging to a specific musical style. The algorithm is based on a stochastic model in which various aspects and qualities of the generated rhythm can be controlled intuitively and in real time. Such qualities are the density of the generated events per bar, the amount of variation in generation, the amount of syncopation, the metrical strength, and of course the meter itself. The kin.rhythmicator software application was developed to implement this algorithm. During a performance with the kin.rhythmicator the user can control all aspects of the performance through descriptive and intuitive graphic controls.

Keywords

automatic music generation, generative, stochastic, metric indispensability, syncopation, Max/MSP, Max4Live

1. INTRODUCTION

In this paper, we propose an approach for real-time rhythm generation based on a stochastic model. This approach contrasts with recent ones involving evolutionary methods such as genetic algorithms [1][2], cultural algorithms [3] or connectionist approaches [4]. In our approach, the algorithm produces a rather static output with slight variations due to the stochastic nature of the algorithm that is characteristic of a certain meter and metrical subdivision level defined by the user. However, the output does not belong to a specific musical style. It is up to the user to modify and control the output of the algorithm during a performance by altering descriptive musical parameters that produce perceivable changes in the output such as the density of events per bar, the amount of syncopation, the degree of metrical strength, the amount of variation in generation, and of course the meter itself. In this sense, the algorithm behaves like a musical companion that responds musically to requests made by the user in musical terms.

kin.rhythmicator is built around two Max/MSP [5] externals (kin.weights and kin.sequencer) that implement the algorithm. It exists as a Max/MSP bpatcher and as a Max4Live [6] device.

2. THE ALGORITHM

The algorithm has two distinct phases. First, the meter entered by the user is subdivided into the number of pulses of a specified metrical subdivision level. Each pulse is assigned a weight value according to its importance in the meter so that a pattern characteristic of the meter emerges. In the second phase, the weight values are used to generate a stochastic performance.

These values are processed and mapped to probabilities of triggering events and their amplitudes in order to enforce or weaken the metrical feel, syncopate according to the specified meter and control the variations in the generated rhythm. The user controls these values indirectly through graphic controls. This gives a very intuitive control over these parameters and over the real-time rhythm generation. In the upcoming sections we describe in detail the steps taken to achieve these results.

2.1 Calculating the Weights

The calculation of the weights of the pulses is articulated in two phases: sorting the pulses by metric indispensability according to Clarence Barlow's metric indispensability formula [7] and calculating the weights based on the stratification levels.

These weights can be thought of as a measure of how much each pulse contributes to the character of the meter. A direct mapping of the weights to probabilities of triggering events gives rise to simple rhythmic patterns expected for the given meter. Variation in the performed rhythms is an innate quality of the algorithm arising from the use of probabilities in the performance.

2.1.1 Sorting by Metric Indispensability

The user inputs meter information in the form of a time signature and a metrical subdivision level which defines the number of pulses the measure is divided into – e.g. a 3/4 meter at the 16th note metrical subdivision level has 12 pulses. Based on this information the meter is stratified by decomposing the number of pulses into prime factors (see Figure 1). Each prime factor describes how each stratification level is subdivided. The stratification level at index 0 is always a whole bar (prime factor 1). Different permutations of the prime factors describe different metrical hierarchies distinguishing this way between simple and compound meters like 3/4 and 6/8 – although they contain the same number of subdivisions at the sixteenth-note level (12) the first is decomposed as 1x3x2x2, while the second as 1x2x3x2.

Barlow´s indispensability [7] takes the prime factors of each stratification level and sorts the pulses in the meter according to how much each pulse contributes to the character of the meter, from the most indispensable to the least important.

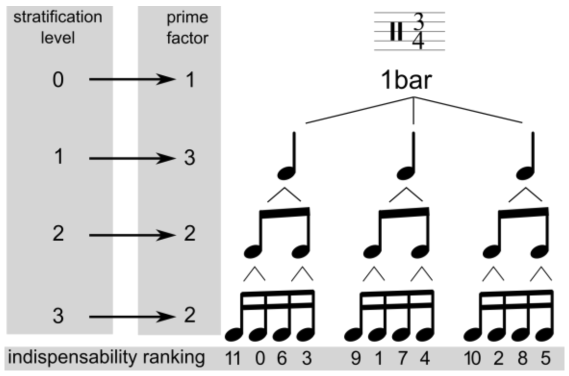

Figure 1. Stratification of a 3/4 meter to the 16th metrical level. At the bottom, the ranking according to Barlow's metric indispensability formula is shown.

2.1.2 *Calculating the Weight Based on the Stratification Level*

We assign to each pulse a weight based on the stratification level it belongs to and its indispensability ranking. Each level i has its own distinct range of weights W_i (see Figure 2):

$$W_i(\max, \min) = (R^{i-1}, R^i) \tag{1}$$

where R is a parameter related to the density of events of the resulted performance and ranges between 0 and 1. Equation (1) implies that the calculation of the ranges begins with the highest stratification level for i = 1 and continues until it reaches the metrical level defined by the user.

The pulse with the highest ranking value in each stratification level, i.e. the most indispensable, is assigned the maximum weight corresponding to the stratification level. The rest of the pulses in the stratification level are assigned smaller weights in the same range following a linear distribution. According to equation (1), for R = 1 all pulses have a weight equal to 1, while for R = 0 only the 1st stratification level survives.

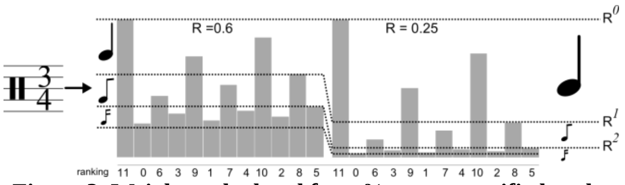

Figure 2. Weights calculated for a ¾ meter stratified to the 16th note level. The ranking of the pulses according to Barlow's formula is indicated below the assigned weights.

2.2 Stochastic Performance

Once the weights of all the pulses are calculated, a performance is generated by cycling through the pulses comprising the metrical cycle and deciding if an event will be triggered in each position or not. During performance, several aspects pertaining its style can be specified, such as the amount of syncopation, the density of events, the metrical strength, the amount of variation, and the events' articulation (staccato or legato).

2.2.1 *Triggering Events*

The probability of triggering an event on a certain time position is derived by the corresponding weight according to a simple exponential relation:

$$p_\ell = n \cdot W_\ell^{\,M} \tag{2}$$

where W_ℓ is the weight assigned previously to pulse ℓ, n is a normalization factor, and M is a user defined parameter related to the metrical feel and ranging between 0 and 1. The above equation functions as a "probability compressor", where for values of M close to 0, the differences in the probabilities are smoothed out, while for values close to 1, the original probabilities arise (see Figure 3).

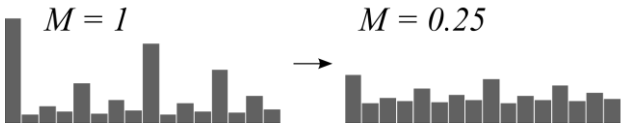

Figure 3. Probabilities are exponentially scaled.

The amplitudes of the triggered events are calculated independently from the probabilities. They are directly proportional to the pulse weights at the strongest metrical feel.

2.2.2 *Generating Syncopation*

Syncopation is introduced in the generated rhythm by "anticipating" pulses in stronger metrical positions. Events are triggered according to the probability assigned to the immediately following next pulse. At the same time, the amplitudes are also anticipated, so that the amplitude of a syncopated pulse sounds louder, thus creating a dynamic accent. The user controls the probability P_S of anticipating a pulse which gives control over the amount of syncopation in the resulted rhythm.

Restrictions are imposed in order for the generated result to be more musical. A mechanism forces syncopation to stop when too many consecutive pulses are anticipated; otherwise for values of P_S close to 1 the resulted rhythm would be just an offset version of the non-syncopated one. An "off-beat" syncopation effect is achieved by resolving consecutive anticipated pulses to the next stressed pulse. Moreover, when only a couple of pulses are anticipated, an event triggered on the following stressed pulse would weaken the feeling of syncopation. In this case the stressed pulse is muted and will not trigger an event, independently from the corresponding probability.

2.2.3 *Controlling Density*

The density of events D refers to how many events are triggered per cycle. On average this is equal to the sum of the probabilities in all pulses:

$$D = \sum_{\ell=\text{all pulses}} p_\ell \tag{3}$$

The density of events and the metrical feel are by nature interrelated. This can be easily seen in extreme cases such as when the density is zero. Zero density means that no events are triggered which is, by definition, a non-metrical state. This degenerate rhythm could belong to any meter and tempo. Similarly, the metrical feel is weakened when events are triggered on every pulse, in other words when the density is maximum, and thus the meter can only be inferred from the amplitudes of the triggered events.

The density of events can be controlled by the parameter R in equation (1). Although the value of R cannot be used as a measure of the actual density of events it serves as an effective way of controlling it without affecting the metrical feel. The probabilities are distributed to the pulses taking into account the stratification level they belong to, preserving the hierarchy and structure of the meter even for low values of R, keeping this way a strong metrical feel when the density is low. On the other hand, the amplitudes of the triggered events are not affected by the changes in the parameter R. This way, when the density reaches its maximum (R = 1) the character of the meter is made evident by the amplitudes of the triggered events.

2.2.4 *Controlling Metrical Strength*

The strength of the metrical feel depends, on the one hand, on the probabilities assigned to the pulses and, on the other hand, on the amplitudes of the generated events. A sense of meter is established when the events are triggered in important pulses (the most indispensable ones). The way the weights are calculated ensures that the more important a pulse is, the more often an event will be triggered in that position and this event will have an higher amplitude accordingly. The more the indispensability relation is preserved among the pulses, the stronger the metrical feel is. When all pulses have similar probabilities of triggering events and the amplitudes of the triggered events are random, not organized and do not establish a pattern, the resulted rhythm sounds random, not belonging to a specific meter.

In order to effectively control the strength of the metrical feel, the probabilities and amplitudes of the triggered events need to be adjusted simultaneously. The probabilities can be directly manipulated through the exponent *M* in equation (2). The normalization factor *n* ensures that the density of events *D* is not affected by the changes in the exponent *M*. In order to weaken the metrical feel as the value of *M* decreases, the amplitudes also get randomized but in a way that the distribution of amplitudes over time is kept constant.

Figure 4 summarizes the main aspects of the performance and their relation to the parameters of the algorithm.

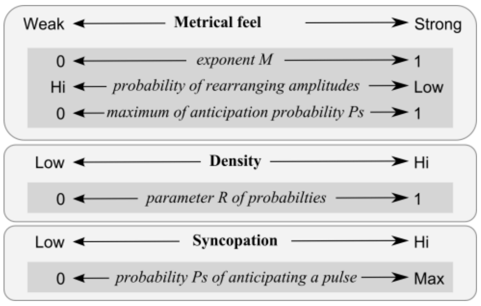
Weak
Metrical feel
Strong
0
exponent M
1
Hi
probability of rearranging amplitudes
Low
0
maximum of anticipation probability Ps
1
Low
Density
Hi
0
parameter R of probabilties
1
Low
Syncopation
Hi
0
probability Ps of anticipating a pulse
Max

Figure 4. A summary of the basic user controls and the corresponding parameters in the algorithm.

2.2.5 *Generating Variation*

The generated rhythm varies and is non-repetitive due to its stochastic nature. The amount and type of variation can be controlled by restricting the mechanisms described above, namely the triggering of events and their syncopation.

At each pulse, two different decisions are made. First, it is decided whether the pulse will anticipate the next one according to the amount of syncopation set by the user. Second, the triggering of an event is decided according to the probability of the corresponding pulse or the following one when anticipating. The variation in the resulted rhythm is controlled by restricting the number of such decisions that are allowed to change from one cycle to the next.

Two modes of variation have been implemented: the stable and the unstable. In the stable mode, the variation revolves around an initial pattern which is randomly generated. In the unstable mode, the rhythm departs from an initial pattern and follows a random walk. It evolves constantly into new patterns. An initial pattern is always generated at the beginning of the performance but the user can re-generate a new random pattern at any time, creating an abrupt change in the performance.

2.2.6 *Events' Articulation*

The duration of the triggered events can be either fixed, in staccato mode, or can extend until the triggering of a new event, in legato mode. Syncopation is enhanced in legato mode by favoring the release of held events on stressed pulses even when no new event is triggered.

2.3 Controlling the Performance: the complexity space

The metrical feel, the amount of variation and the amount of syncopation form what we call a "space of complexity". A rhythm is considered to be simple, when the metrical feel is strong, variation is kept to a minimum and there is no syncopation. On the other hand, when the metrical feel is weak or when syncopation is introduced into the rhythm or when the rhythm is constantly changing, then the rhythm is perceived to be more complex. Rhythmic complexity in this sense is attributed to combinations of different aspects of the rhythm: metrical strength, syncopation and variation.

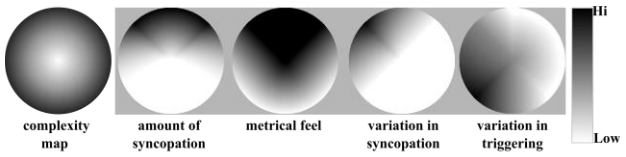

Figure 5. Contour plot of the functions used in the complexity plane to map position coordinates to the parameters of the algorithm. At the left side a contour of the expected complexity of the generated rhythms is shown.

We grouped the parameters of the algorithm related to complexity into a two-dimensional map (see Figure 5). As one moves away from the center the resulted rhythm becomes more complex. The dependence of each parameter on the position in the complexity map was empirically set, taking into consideration some basic restrictions derived from the nature of these parameters and our experience with various settings of the algorithm. Some of these restrictions are: i) when the metrical feel is low, syncopation is meaningless, ii) variation in the syncopation decisions apply only when the amount of syncopation is above a certain value, iii) when the amount of syncopation is significant the syncopation feeling is weakened by too much variation in the triggering decisions.

3. APPLICATIONS

3.1 Max/MSP Externals

The algorithm was implemented as two Max/MSP externals. Several other externals and abstractions have been developed in order to facilitate the use and implementation of the algorithm into Max/MSP applications. All externals and abstractions are completely cross platform, Windows and Mac OS.

The first phase of the algorithm, namely the generation of weights, is performed by the kin.weights external. The parameter *R* of equation (1), which controls the density of events, is directly fed into the external as a floating-point number in the range [0, 1].

The second phase, the triggering of events based on the parameters mentioned in the previous section is performed mainly by the kin.sequencer external. The weights calculated by kin.weights are fed into kin.sequencer which generates a performance by cycling through each pulse comprising the metrical cycle and deciding if an event will be triggered in that position or not. The amount of syncopation, the metrical strength, and the amount of variation can be controlled by respective messages to the external.

A java script suited for the jsui Max/MSP object was developed to visualize and improve user interaction with the complexity space described in 2.3.

3.2 The kin.rhythmicator bpatcher

The kin.rhythmicator Max/MSP bpatcher abstraction was built around the above externals. It is intended to be used in Max/MSP based applications and installations which implement some kind of rhythmic interaction. Such installations can take the form of virtual musical instruments, compositional tools or interactive installations. It is easily integrated into Max/MSP patches. It can be controlled by various devices, from simple MIDI controllers to complex game controllers and is ready to directly trigger sound on any MIDI enabled synthesizer.

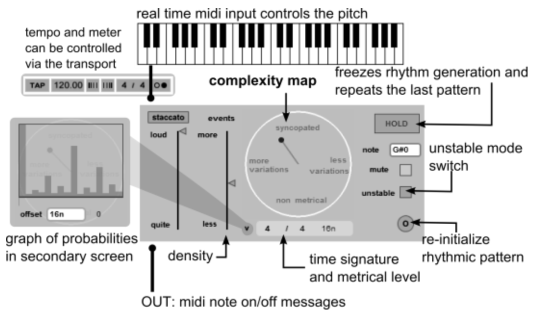
real time midi input controls the pitch
tempo and meter can be controlled via the transport
complexity map
freezes rhythm generation and repeats the last pattern
unstable mode switch
graph of probabilities in secondary screen
density
time signature and metrical level
re-initialize rhythmic pattern
OUT: midi note on/off messages

Figure 6. The interface of the kin.rhythmicator application.

The abstraction implements all the features of the algorithm and has a compact user interface when loaded into a bpatcher object (see Figure 6). All parameters of the algorithm can be set during performance directly on the user interface, as messages, or through the pattr system for storing preset files in Max/MSP.

Single notes or chords can be fed to kin.rhythmicator in real time making it follow a melody or a chord progression which can be either pre-scheduled, performed in real time by a musician or generated by some generative or analysis algorithm.

Several instances of the abstraction can be loaded at the same time for generating several rhythmic layers. All instances can be synchronized by the Max global transport. Also, one can generate polyrhythms by synchronizing different instances of kin.rhythmicator with different time signatures to the global transport.

3.3 The Max4Live MIDI Device

We developed a Max4Live device that can be used as a compositional and/or performance tool to dynamically generate rhythms. All parameters can be controlled through MIDI, automated with envelopes and saved together with the Live Set.

The device is built as a MIDI FX device, which means it can be loaded into a Live's MIDI track. It can be used alongside VST or the Live’s instruments. More than one instance can be loaded in the same or different MIDI tracks. The interface is very similar to the Max/MSP bpatcher abstraction described above (see Figure 6).

The kin.rhythmicator max4Live device reads automatically the time signature and the play position of the Live transport and follows any time signature change in the song, so that there is no need to explicitly set the time signature on each kin.rhythmicator instance. All instances of the device are in sync with the rest of the Live Set. An offset parameter allows for a phase difference between each kin.rhythmicator and the global transport.

Two MIDI modes of operation have been implemented: thru and listening. In thru mode, the MIDI input is forwarded directly to the output without being changed. The generated rhythm is output as MIDI note on/off messages according to the MIDI note set on the kin.rhythmicator. In listening mode the rhythm generated follows the melody or chord progression at the input of the device.

4. CONCLUSION AND FUTURE WORK

The algorithm and applications introduced here present a novel approach to automatic rhythm generation. Departing from a preexisting metrical template containing the time signature and metrical weight distribution, and the subdivision level, a user can specify a performance controlling several musical parameters. Instead of specifying in detail the rhythmic parts and variations needed in a musical composition or performance, one can use kin.rhythmicator devices to control parts or the whole of the rhythmic section. These parts can be thought of as constrained improvisations that take the place of a detailed music score.

The real time and intuitive character of the controls and performance of the kin.rhythmicator helps in creating music more responsive to user actions. Controlling the metrical strength and density of events effectively has been made possible by taking into account the hierarchical structure of the meter in mapping the output of Barlow´s indispensability formula to the probabilities. A syncopation algorithm based on the anticipation of pulses that tends to keep a strong metrical feel is introduced.

Future development of the kin.rhythmicator algorithm and devices include the development of intelligent agents, which collaborate in generating a coherent output.

The kin.rhythmicator Max/MSP application and Max4Live device are available for download at our group website: http://smc.inescporto.pt/kinetic/

5. ACKNOWLEDGMENTS

This research was done as part of the project “Kinetic controller driven adaptive music composition systems”, (ref. UTAustin/CD/0052/2008), supported by the Portuguese Foundation for Science and Technology for the UT Austin| Portugal partnership in Digital Media.

6. REFERENCES

[1] Bernardes, G., Guedes, C., Pennycook, B. “Style emulation of drum patterns by means of evolutionary methods and statistical analysis.” *Proceedings of the Sound and Music Conference,* Barcelona, Spain, 2010.

[2] Eigenfeldt, A. “The Evolution of Evolutionary Software Intelligent Rhythm Generation in Kinetic Engine.” *Proceedings of EvoMusArt 09, the European Conference on Evolutionary Computing*, Tübingen, Germany, 2009

[3] Martins, A. and Miranda, E. “Breeding rhythms with artificial life.” *Proceedings of the Sound and Music Conference,* Berlin, Germany, 2008.

[4] Martins, A. and Miranda, E. “A connectionist architecture for the evolution of rhythms.” *Proceedings of EvoWorkshops 2006 Lecture Notes in Computer Science*, Berlin: Springer-Verlag, Budapest, Hungary, 2006

[5] http://cycling74.com/

[6] http://www.ableton.com/maxforlive

[7] Barlow, C. “Two essays on theory”. *Computer Music Journal*, 11, 44-60, 1987