Generation and control of automatic rhythmic performances in Max/MSP

George Sioros, Carlos Guedes

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

Abstract. We present two algorithms and the corresponding software applications for automatically generate music rhythms and rhythmic variation. They can be used to substitute static loops with constantly varying rhythms which are intuitively managed and controlled in real time. The first, the kin.rhythmicator, generates rhythms characteristic of a certain rhythm and that do not belong to a specific musical style. It 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 second, the kin.recombinator, recombines in real time a batch of MIDI drum loops in order to get non-excessively repetitive combinations of loops during a performance. The algorithm's power lies in a novel analysis of the drum loops which sorts them in increasing order of complexity prior to utilizing them in the recombination algorithm. This way, the user can effectively control the complexity and variation in the resulted rhythm during performance.

Keywords: automatic music generation, generative, stochastic, metric indispensability, syncopation, recombination, MIDI, Max/MSP, Max4Live

1 Introduction

Devising different strategies for generating rhythm in real time is one of the goals of the [will name it later] project. After proposing the use of Genetic Algorithms [1], we propose two new algorithms for real-time rhythm generation. The first, called thee kin.rhythmicator, is based on a stochastic model. The second, called the kin.recombinator, is essentially a simple yet effective method for recombining MIDI drum loops of a certain style (such as those available in Apple's GarageBand).

The kin.rhythmicator 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.

In the kin.recombinator application, the user can recombine in real time, with different degrees of complexity, a batch of MIDI drum loops in order to get non-excessively repetitive combinations of loops during a performance. The user can control the amount of variation in recombination during performance, as well as different degrees of complexity.

kin.rhythmicator and kin.recombinator are implemented as Max/MSP [5] externals and java classes around which Max/MSP applications and Max4Live [6] devices were built.

2 Automatic Rhythm Performance

2.1 The kin.Rhythmicator

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.

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.

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 Fig. 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).

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.

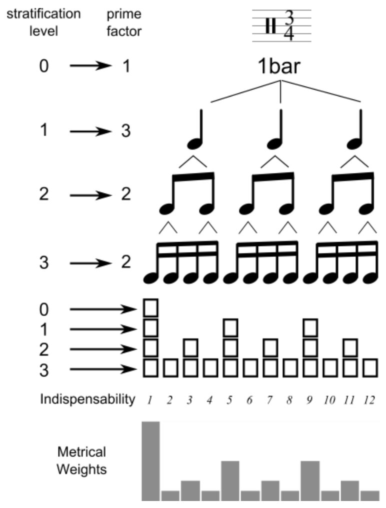

Fig. 1. Stratification of a 3/4 meter to the 16th metrical level. At the bottom, the ranking according to Barlow's metric indispensability formula and the metrical weights assigned to each pulse are shown

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 Wi (see Fig. 1):

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

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.

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).

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 (see Fig. 2):

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

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.

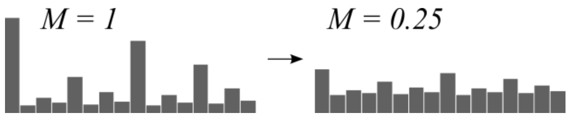

Fig. 2. Probabilities are exponentially scaled.

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.

Generating Syncopation. Syncopation is introduced by "shifting" parts of the calculated probabilities and amplitudes patterns (see Fig. 1 and Fig. 2) by one pulse to the left. This results in "anticipating" pulses in stronger metrical positions. The user controls the amount of syncopation in the resulted rhythm by controlling how long these portions are and how often such a shifting occurs through the global probability P_S of syncopating a pulse.

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 high syncopation values the resulted rhythm would be just an offset version of the non-syncopated one. An "off-beat" syncopation effect is achieved by choosing the shifted portions to end on stressed beats.

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 can be controlled by the parameter R in equation (**Error! Reference source not found.**). 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.

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. 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 that this event will have an accordingly higher amplitude. The more the indispensability relation is preserved among the pulses, the stronger the metrical feel is.

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.

Fig. 3 summarizes the main aspects of the performance and their relation to the parameters of the algorithm.

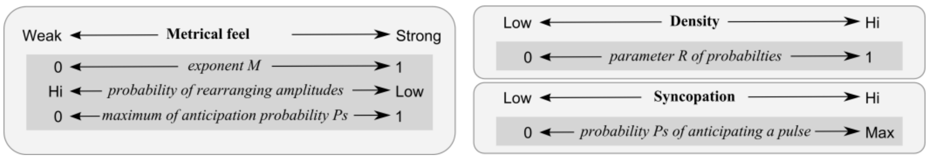
Weak
Metrical feel
Strong
0
exponent M
1
Hi
probability of rearranging amplitudes
Low
0
maximum of anticipation probability Ps
1
Low
Density
Hi
0
parameter R of probabilties
1
Low
Syncopation
Hi
0
probability Ps of anticipating a pulse
Max

Fig. 3. A summary of the basic user controls and the corresponding parameters in the algorithm.

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.

Event's 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.

Controlling the Performance: the Complexity Map. 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.

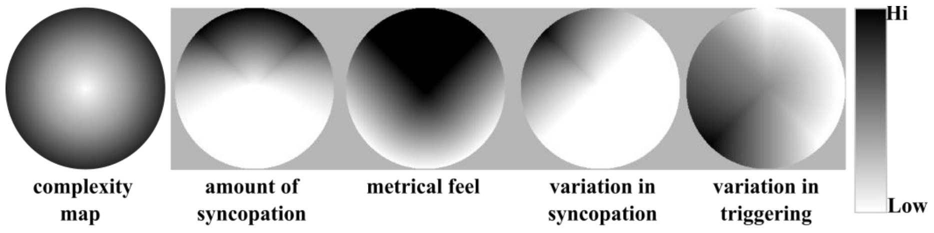

Fig. 4. Contour plot of the functions used in the complexity plane to map position coordinates to the parameters of the algorithm. At the left side a contour of the expected complexity of the generated rhythms is shown.

We grouped the parameters of the algorithm related to complexity into a two-dimensional map (see Fig. 4). 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 basic restrictions derived from the nature of these parameters and our experience with various settings of the algorithm.

2.2 The kin.Recombinator

Recombinance is an effective technique to generate music according to a certain style [8]. The kin.recombinator application generates rhythmic patterns by recombining existing ones. The recombination process consists of playing back MIDI drum loop files by selecting portions of these files at regular intervals. An analysis of the files is performed prior to the recombination, in order to sort them according to their complexity and, in this way, better control the resulting rhythms.

The algorithm can be divided in two phases. In the first phase a set of MIDI drum loop files input by the user are analyzed and sorted according to how complex they are, from the simplest to the most complex. This complexity measure is based on a new method for measuring syncopation, by comparing the patterns against a template characteristic of their meter.

Calculation of the complexity scores and sorting of MIDI drum loops. Various approaches for measuring complexity in rhythmic patterns exist, such as pattern matching techniques [9], rhythmic syncopation measures [10] and analysis of the mathematical or geometrical properties of the patterns [10, 11]. Here, we define a new one, which is based on the same principle as G. Toussaint's *metric complexity* [10], which is a comparison of a rhythmic pattern against a template characteristic of its meter. Unlike Toussaint's approach that uses the template as a way of calculating the metrical weights, we use the template as the fundamental tool for analyzing and defining relationships between the events comprising the pattern. Moreover, unlike most methods for measuring complexity, which use a binary representation of the patterns and ignore the amplitudes of the events that comprise the pattern, we take into account the relative amplitudes of the events in our calculation.

The user provides rhythmic patterns in the form of a collection of MIDI files. The MIDI files are read and are quantized according to a fixed quantization grid. This way each measure in a pattern is subdivided into pulses according to the time signature and the quantization grid. The quantization grid value we use is the 32nd note. An amplitude value is assigned to each pulse, according to the MIDI velocities found at that time position after the quantization. All MIDI files should have the same time signature and bra-length.

A metrical template, such as the one described by F. Lerdahl and R. Jackendoff [12], that defines metrical hierarchy is constructed by stratifying the meter found in the MIDI files. The meter is stratified into metrical levels in a hierarchical manner so that each pulse belongs to a specific metrical level and all lower ones. The stratification process is the same as the one described for the kin.Rhythimcator in 2.1. A metrical weight value is calculated and assigned to each pulse according to equation (1) for $R = 0.5$. Each rhythmic pattern found in the MIDI files is compared against the metrical template yielding a separate score for each pulse in the pattern. The result is further filtered by the aforementioned values of the metrical weights. The calculated scores can be thought of as a measure of how much each pulse contradicts the metrical structure described by the template and, in this sense, how much each pulse contributes to the syncopation of the pattern. Finally, a complexity score is assigned to each MIDI file taking into account, in addition to the syncopation, the density of MIDI events in each file.

Comparing a Rhythmic Patterns to the metrical Template. The comparison of a rhythmic pattern to a metrical template is essentially a process of spotting the pulses in the rhythmic pattern that contradict the prevailing meter described by the template. In that sense, these pulses are mainly responsible for any syncopation present in the rhythmic pattern and the result of the comparison is a measure of that syncopation. Eliminating the events that occur regularly on the beat helps distinguishing the syncopating pulses. Such events do not generate syncopation. In order to define how each pulse contributes to a steady beat, its relation to its neighbor pulses must be examined. A score is calculated as the average difference of the amplitude of the pulse under consideration from the amplitudes of the neighbor pulses in each metrical level. Negative differences are always set equal to zero. A low score signifies that the pulse contributes to a steady beat and therefore does not contradict the meter.

The various pulses in a rhythmic pattern have a different potential in contradicting the prevailing meter, or syncopating, depending on which metrical levels they belong to. This syncopation potential can be thought of as the opposite of the metrical weights, which essentially is the potential of a pulse to contribute to a steady beat. The scores calculated above represent how much a pulse contradicts the meter with respect to its relation to its neighbors but do not take into account this syncopation potential. We therefore multiply the scores previously calculated by a factor proportional to the inverse of the metrical weights of equation (1). This way a pulse that belongs to metrical level 0 could never contradict the meter, irrelevant of if an event exists in this or any other pulses of the meter. Of course, the absence of an event in a high level pulse creates the possibility for an event in some other pulse, probably of a low metrical level, to produce a strong syncopation, but this is reflected on the syncopation potential of the low metrical level pulse.

The last step taken in the calculation of our syncopation measure is to sum the scores of all pulses in the pattern and normalize the result. Normalization is performed by dividing the result by the maximum possible sum for the meter and bar-length. This maximum is calculated by comparing against the metrical template a pattern in which all pulses of the lowest metrical level have maximum amplitudes and all other pulses have zero amplitude.

Calculating the complexity of a MIDI file. The complexity of a drum loop can be thought of as a vector in a two dimensional space, where one dimension is the density of the events and the other is the syncopation. The length of the vector is the complexity score. The syncopation measure is already normalized and can be directly used as one of the coordinates of the vector. The density of events is calculated as the sum of all the MIDI velocities found in the MIDI file. This sum represents an effective density, since it does not correspond to the number of events in the pattern. This number needs to be normalized before it can be used as a coordinate in our complexity plane. We normalize the density by dividing with the largest density value in the collection of MIDI files provided by the user. The total complexity of a file is then calculated as:

$$Complexity = \sqrt{density^2 + syncopation^2} \tag{4}$$

This is the value used to sort the MIDI files, from the most simple to the most complex.

Recombining the rhythmic patterns. The sorted MIDI files form a two dimensional space, where the vertical dimension represents their order of complexity and the horizontal represents their evolution in time (see Fig. 5). All files share the same time signature and have the same bar-length and, therefore, are perfectly aligned.

A global transport controls the current playback position which is common to all files. The tempo of each file is ignored and the playback follows the transport's tempo, controlled in real time by the user. Playback is performed in a loop. At every beat, a new file is randomly selected and playback continues in this file at the current transport's position (see Fig. 5), preserving always the metrical position.

Instead of selecting a file out of the whole collection of the provided MIDI files, the selection process is restricted to a smaller collection of files. During the performance, the user controls the resulted rhythm by controlling in real time the range of files, from the simplest to the most complex that can be selected for playback. Increasing the range leads to more variation in the resulted rhythm, while moving the entire range vertically to more or less complex patterns controls the complexity of the rhythm.

The output of the recombination algorithm undoubtedly depends on the provided MIDI files. In order for the output to be coherent, all files should belong to the same music style. When the files have a similar structure, either a large scale structure consisting of several bars, or at the beat level, this structure will be reflected also in the outcome of the recombination. This comes about as a direct result from using a global transport to control playback.

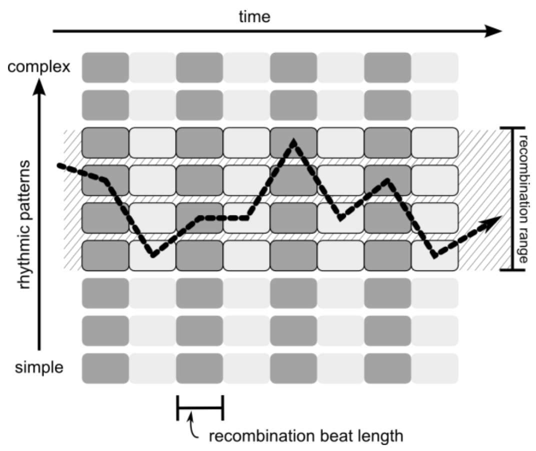

Fig. 5. The rhythmic patterns, sorted from simple to complex, are selected for playback at regular intervals according to the recombination range.

3 Applications

3.1 The kin.Rhethmicator Max/MSP Application

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 csclulation of weights is performed by the kin.weights external. The weights 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 2.1.

The kin.rhythmicator Max/MSP 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.

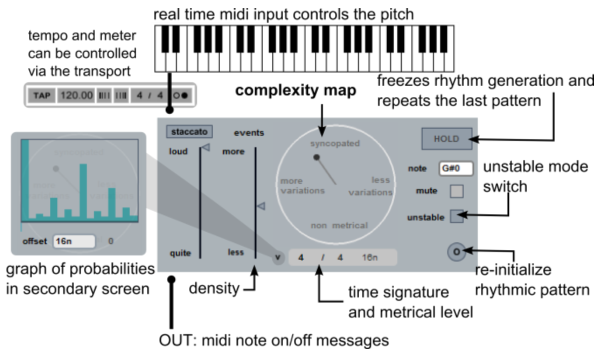
real time midi input controls the pitch
tempo and meter can be controlled via the transport
complexity map
freezes rhythm generation and repeats the last pattern
unstable mode switch
re-initialize rhythmic pattern
graph of probabilities in secondary screen
density
time signature and metrical level
OUT: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

Fig. 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 Fig. 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.

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 Fig. 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.

3.2 The kin.Recombinator Application

The algorithm has been implemented as the *kin.recombinator* Max/MSP application. A collection of Max/MSP externals, java classes suitable to be loaded to the mxj Max/MSP object and Max/MSP abstractions were developed as parts of the application.

The user drags and drops a folder containing MIDI files into the Max/MSP application. The files are automatically sorted and the global Max/MSP transport controls playback. The user can graphically control the range of files being recombined at any one time with a range slider like the one in Fig. 7. The MIDI files are read and quantized to the 32nd note level by the java class *kinMIDIFileReader*. Rhythmic patterns are constructed in the form of lists of amplitudes and are passed together with the respective time signatures to a subpatch where the effective density and syncopation score are calculated by the *kin.OffBeatDetector* Max/MSP external. The score is then stored in a collection object. After finishing calculating the scores for all the files, the files are sorted according to their scores. The *kin.RecombineMIDIFiles* abstraction is performing the playback. It selects at every beat a new file for playback which is read by the *kin.MIDIFileReader*.

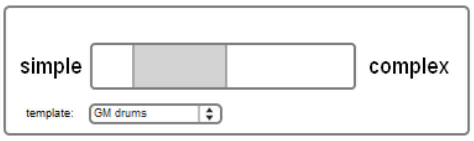

Fig. 7. The kin.recombinator user interface. The user drags and drops a folder containing MIDI loops anywhere in the application’s window. During the performance the complexity and amount of variation can be controlled graphically with a range slider.

The Max/MSP applicationS and Max4Live devices are available for download at our group website: http://smc.inescporto.pt/kinetic/

4 Acknowledgements

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.

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
8. D. Cope: *Experiments in Musical Intelligence*, Middleton, A-R Editions, 1996
9. J. Pressing: "Cognitive complexity and the structure of musical patterns", *Proceedings of the 4th Conference of the Australian Cognitive Science Society*, Newcastle, Australia, 1997
10. G. T. Toussaint: "A mathematical analysis of African, Brazilian, and Cuban clave rhythms", *Proceedings of Bridges: Mathematical Connections in Art, Music, and Science,* Towson University, Baltimore, Maryland, July 27-29, 2002
11. F. Gómez, A. Melvin, D. Rappaport, and G. Toussaint: "Mathematical measures of syncopation", *In BRIDGES: Mathematical Connections in Art, Music and Science*, p. 73–84, Jul 2005.
12. F. Lerdahl & R. Jackendoff: *A Generative Theory of Tonal Music*, Cambridge, The MIT Press, 1996