The Traveling Percussionist

Phil Lopes and Paulo Urbano

LabMAg - Faculdade de Ciências da Universidade de Lisboa,
Campo Grande 1749-016, Lisboa, Portugal
louisphil.lopes@gmail.com,
pub@di.fc.ul.pt
http://bookmark.labmag.di.fc.ul.pt

Abstract. In this paper we introduce the concept of a progressive percussion graph as a musical space and the metaphor of composition as the musical expression of a traveling experience in that space. A Progressive Percussion Graph is a directed graph where each node is associated with a particular percussion rhythm and each connection corresponds to a rhythmic progression, generated through optimization processes, from one percussion rhythm to another, respecting the connections direction. We have explored different optimization techniques and different path-finding algorithms resulting in a rich and diverse musical output.

Keywords: Generative Music, State Space Search, Optimization, Sound Morphing.

1 Introduction

This work describes a form of hybrid human-generative percussion rhythmic composition, that can be viewed as a walk along **Progressive Percussion Graphs**. A Progressive Percussion Graph is a directed graph where each node is associated with a particular percussion rhythm and each connection corresponds to a rhythmic progression from one percussion rhythm to another, respecting the connection's direction. The variety of ways of traveling the graph offers the user multiple compositional options.

The user will be responsible for the percussion graph structure: setting the nodes and the directed connections, and associating percussion rhythms to every node. The rhythmic progressions associated with connections, from one node to the other, will be the rhythmic traces of an optimization computational process where one rhythm (associated with the connection input node), being the starting point, is subject to gradual transformations until converging towards the goal rhythm (associated with the connection output node). The nature of the rhythmic progressions will depend on the optimization algorithm chosen and its respective parameters, affecting both the length of the rhythmic progression and the progression dynamics. This concept is most commonly known as sound morphing [18]. The user can edit connections using specific optimization filters, like for example, deleting some sub-sequences that repeat or cutting sub-parts that do not progress in a satisfactory way.

© Springer-Verlag Berlin Heidelberg 2012

Musical composition can be viewed metaphorically as the expression of a cinematic experience on a progressive percussion graph: a musical voyage through the graph nodes and connections. The exploration of the graph can be just a musical "promenade" with no specific destination goal playing the connections along its way. Alternatively the musical traveller can be oriented towards a desired rhythm and the resulting composition (sequence of connections) being the solution to a path finding problem. To solve such a problem we can apply several standard path finding algorithms with different properties that generate a vast variety of compositions. It is also possible to create an analogy with the traveling salesman problem and generate compositions that correspond to a tour that visits all nodes exactly once.

We have developed a prototype named Percussion Walker where the human composer can create percussion graphs, can associate percussion rhythms with each node of a particular graph and can choose between three optimizers (genetic algorithm, hill-climbing and stochastic hill-climbing) for generating automatically a progressive percussion graph. We have created three types of filters for modifying the optimizer's output sequences of rhythms. Picking a particular progressive percussion graph the user may choose to create a rhythm piece that starts in a particular node and ends in a goal node, using several standard algorithms [15] for searching the graph. It is also possible to make a traveling salesman type of percussion piece, after the graph expands into a complete one, using a known local search algorithm (2-opt) [4], starting with an initial circuit using the nearest neighbor algorithm. The Percussion Walker Prototype can be seen in action here [13].

The paper is organized in the following way: in section 2 we will discuss related work. Section 3 we will describe the concept of a percussion graph, while in section 4 we will look at a particular type of Percussion Graph: the Progressive Percussion graph. We will present the similarity metric that measures the similarity between a rhythm and a target rhythm and which is at the core of the optimization process used for filling graph connections with sequences of percussion rhythms. We will examine the application of three optimization algorithms: genetic algorithms, hill-climbing and stochastic hill-climbing, and will close the section presenting three optimization filters. Section 5 will be dedicated to rhythmic composition viewed as a path finding search, and finish in section 6, pointing future work.

2 Related Work

Evolutionary Computation has been used widely used in music [11], for example, in composition, sound synthesis, improvisation and expressive performance analysis. One of the early examples of Evolutionary Computation applied for music is Horner and Goldberg [7], where a Genetic Algorithm was used to perform thematic bridging. GenJam by Biles, John A. [1], is another type of evolutionary system in which its objective was to model a novice jazz musician, who is learning how to improvise and perform live shows. The GenJam works by using two types of populations one for measures and one for phrases as a way to build a solo. An individual in the measure population maps to sequences of MIDI events, while an individual in the phrase population maps to indexes of measures in the measure population. This makes it so that there is not just one single best measure or phrase. Achieving the perfect solo wasn't really the point of this work, but more of a way where the GenJam can apply various melodic ideas to any tune. In 1997 Brad Johanson and Riccardo Poli developed the GP-Music System [9] that used genetic programming to breed melodies according to both human and automated ratings. Phon-Amnuaisuk et al. [14] used a GA to generate traditional musical harmony, i.e. chord progressions where they used a fitness function derived from music theory with surprisingly successful results. The Swedish composer Palle Dahlstedt presents an evolutionary system which can create complete piano pieces using a recursive binary tree representation combined with formalized fitness criteria [5]. The NEAT Drummer [6], automatically provides drums to accompany pre-existing music using neuronal networks and interactive evolution. Sound Morphing has also been extensively researched by Wooller R. [18], such as its favorable usage in electronic dance music [19] or as a way of easily integrating the user using the morph table interface [2]. Although this work uses morphing extensively to generate pattern sequences, our method relies more heavily on the genetic algorithm model, rather then the probabilistic approach.

The main objective of Horowitz's system [8] was to generate automated rhythms through a genetic algorithmic process. Users would then influence this genetic process by evaluating each generated rhythm with a "like or "dislike. Even though the system is using an interactive genetic algorithm approach [16] the subjectivity and variability of a users criteria can sometimes be ambiguous. For that matter, user evaluation wasn't the only determining factor for the fitness function. Objective functions, containing various rhythmic rules (such as density, beat repetition, etc.) would also influence and steer the evolutionary process into a more specific direction.

CONGA [17] is another type of evolutionary music composition system developed by Tokui & Ida. However, unlike Horowitz's system, the CONGA system relies on both the Genetic Algorithm and Genetic Programming [10] concepts for generating new rhythmic solutions, which had been previously proven successful [3][1][9]. The CONGA system was designed taking into consideration three main aspects, the search domain, the genetic representation and the fitness evaluation. The search domain consisted of 4 to 16 measure rhythmic patterns (sequence of notes). The genetic representation combined the genetic algorithm (GA) and genetic programming (GP) approaches, where GA individuals represented short pieces of rhythmic patterns, while the GP expressed how these patterns were arranged in the musical structure. The Fitness Function consisted of a user evaluation approach, accompanied by an evaluation assistant module.

Yee-King's Evolving Drum Machine [20], is a good example of how the evolutionary and generative process can be taken into consideration for the music creation process. In this system the user can simply input a target sound (or rhythm), which serves as the rhythmic objective or the goal of the evolutionary process.

3 The Percussion Graph

A percussion graph is a directed graph where each node is associated with a percussion rhythm (which we will simply call a rhythm) and each connection is associated with an ordered non-empty sequence of rhythms. These connections also have costs that correspond to the number of rhythms in their respective sequences. The rhythms of a percussion graph are abstract enough to be defined however we want, as there are no constraints imposed on the musical content. The association of nodes with a rhythm as well as the association between a connection and its sequence of rhythms do not imply any sort of commitment to any form of symbolic or computational representation.

A rhythm will consist of a set of notes. Each note like the name entails, is a musical note that will be played by a specific instrument at a specific point in time. It is also important to note that each rhythm will have a constant tempo and won't be taken into consideration

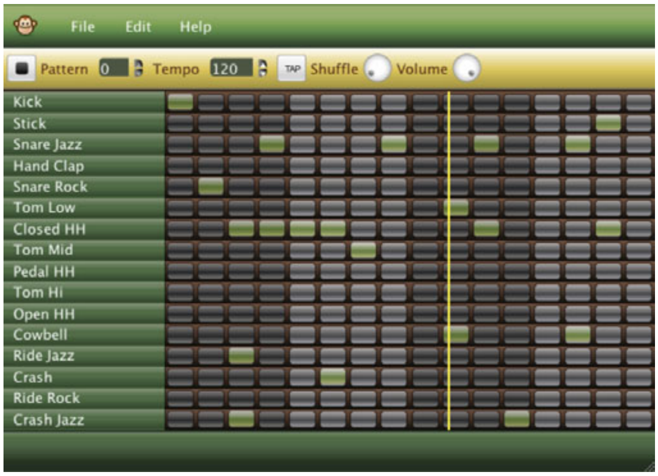

Fig. 1. An Example of a Rhythm of a Percussion Graph (Image taken from the Monkey Machine [12])

For this research we have used a particular type of percussion rhythm, which we explain below, however it is important to note that a rhythm within the percussion graph concept, can be linked to any other type of percussion rhythm.

The rhythm we have used consists of a 16x16 grid (see Figure 1), where each line of the grid represents a percussion instrument and each column represents the time in which the instrument is played. The timeline is a standard 4/4 notation, divided in eight notes, this means that the instrument can be played up to 16 times in a measure, however the drum machine was limited to a single inputted measure. Even though each rhythm can be played at different tempos, we have chosen to to keep the tempo homogeneous for all the rhythms within the a percussion graph.

Velocity is the force in which an instrument is played (in the MIDI standard), and we have considered two types of velocity note. The active note (ex. playing the instrument) and the silent note (ex. not playing the instrument).

4 Progressive Percussion Graphs

As previously stated in the percussion graph concept there are no restrictions regarding the association between graph components and rhythms: a node can be paired with any kind of percussion rhythm and a connection can be associated with any possible sequence of rhythms. The only constraint is that each rhythm must be of a particular type.

For this research however, we have envisioned a particular type of hybrid percussion graph, the **Progressive Percussion Graph**, involving collaboration between a human and the computer. The human percussion graph composer defines both the structure of the intended graph and the rhythms associated with the nodes, but the sequences of each connection are automatically generated, through an optimization process, corresponding to progressive transformations between the two correspondent linked nodes.

The optimization problem is the following: starting from a particular rhythm, how is it possible to reach the final rhythm in the minimal number of steps, where a single step is always the transformation of a rhythm into another. The trace left by the optimization process will be a sequence of percussion rhythms, which hopefully will correspond to a progressive transformation from a connection's start node rhythm to it's end node rhythm. By traveling along the connection's rhythmic sequence, the rhythms will progressively be more similar to the end node's rhythm and less similar to the starting rhythm. The smoothness of this progression will depend on the single step transformations and the distance towards the final rhythm may not always decrease as the optimization process may be unstable. The intensity of a single step transformation of a rhythm into another (difference between them) depends on the operator used by the optimizers and is subject to variation and control.

It is also necessary to define a way to stop the optimization process as it might not be in the interest of the user to arrive exactly at the goal rhythm but when a rhythm is sufficiently near the goal rhythm (the similarity towards the goal is above or equal to a user defined satisfaction threshold). It was also necessary to create a fail safe in the cases where the generative process could take too long to converge towards the objective. Thus the optimization might end when one of these conditions holds: similarity towards the goal rhythm is above the satisfaction threshold or the sequence length is bigger than a certain size limit.

In order to guide any optimization process we must be able to measure the similarity between any two percussion rhythms. The chosen similarity metric will be highly sensible to the type of rhythms used in the percussion graph. In the following section we present, a particular similarity metric adapted to the rhythm type used in our research, that were described in the last paragraph of section 2.

4.1 Measuring Similarity

For us it was important to consider two main criteria in measuring the similarity between a rhythm and a target rhythm. First it was necessary to take into account the nature of notes in both rhythms, the number of notes that are coincident in time, instrument or both - velocity was not a main concern however, because all of the notes are played with a constant velocity. Secondly, it was also important to take into account the similarity between two rhythms in terms of the number of played notes, independently of their nature. The similarity function is a weighted sum of a **Notes Number** similarity function and a **Notes Similarity** function, and ranges from a value of 1 to 10 (10 when the two rhythms are exactly the same).

$$Similar(r, r_t) = \frac{Sim_{nn}(r1, r2) \times W_{nn}}{W_{nn} + W_n} + \frac{Sim_n(r, r_t) \times W_n}{W_{nn} + W_n} \tag{1}$$

Notes Number Similarity. The name Notes Number refers to the number of notes that are played within a single rhythm, for example if a rhythm has X playing notes the note number will be X. The note number similarity between any two rhythms will depend on their number's absolute difference.

$$Sim_{nn}(r, r_t) = 10 \times \frac{1}{(|(NoteNumber(r_t) - NoteNumber(r)| + 1)} \tag{2}$$

Notes Similarity. The Notes Similarity evaluation will compare one rhythm (r) against a target rhythm (r_t), in terms of the nature (instrument and time) of their notes. To measure note similarity we consider three aspects: Common Instruments, Common Times and Common Notes.

- **Common Instruments (I):** The number of distinct instruments that are played in both rhythms. Note that, if r has two active notes that play maracas and r_t only has one active note with that instrument, that instrument will only be counted once.
- **Common Times (T):** The number of distinct times that are played in both rhythms. Repeated active notes with the same time will be counted in the same fashion as in the Common Instruments.
- **Common Notes (N):** The number of active notes that appear in both Rhythms.

Each of these aspects will correspond to a respective function: I(r,r_t), T(r,r_t) and N(r,r_t). The Notes Similarity function will be normalization of the weighted sum of these three functions and again will be a value between 1 and 10.

$$Sim_n(r, r_t) = 10 \times \frac{I(r, r_t) \times W_I + T(r, r_t) \times W_T + N(r, r_t) \times W_N}{numI(r_t) \times W_I + numT(r_t) \times W_T + NoteNumber(r_t) \times W_N} \tag{3}$$

4.2 Optimization Process

Hill Climbing. Hill climbing is a mathematical optimization technique, belonging to the family of local search and is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution using a neighborhood operator. As soon as a change produces a better solution (best-first), an incremental change is made to the new solution, repeating until no further improvements can be found. The optimization trace that will be associated with the respective connection will be the consecutive solutions obtained through the incremental changes that the starting node rhythm is subject until the stop condition is fulfilled (a limit sequence is reached or the similarity towards the connection output node reached a satisfactory threshold). Hill-Climbing does only guarantee local optima but it is not a problem in a musical setting.

We defined a special rhythm modification function (neighborhood function) that allowed us to modify a previous rhythm through the process of randomly adding, removing and replacing a number of N notes. The N parameter is important for achieving a more or less smooth rhythmic progression towards the goal rhythm. A higher value of N will imply a shorter progression (lower values of N will result in longer progressions). This allowed us to constantly modify the last inserted rhythm of a connection, which could then be evaluated through our similarity function.

Note that the resulting rhythmic sequence will be strictly monotonic: the similarity towards the goal rhythm will always increase.

Stochastic Hill Climbing. The stochastic hill climbing process differs specifically in that the solution that is inserted into the connection depends on a probability, rather then the best first approach. The probabilistic formula favors better solutions, however there is always a slight probability that a worse solution might be inserted into a connection. So instead of a steady progression like the normal hill climbing, the stochastic has a chance of regressing in rhythm similarity.

The probability of acceptance depends heavily on the difference of merit between the score obtained by the last solution (v_c) and the score of the new solution (v_n) and of an additional parameter T. The T parameter influences the sway of the acceptance rate. The higher the T value: the higher the chance of accepting a worse solution.

$$p = \frac{1}{1 + e^{\frac{eval(v_c) - eval(v_n)}{T}}} \tag{4}$$

Genetic Algorithm. For the Genetic Algorithm optimization, each Chromosome (or Individual) will represent a percussion rhythm and will be an array of bits (where 1 represents an active note and 0 an inactive note). The fitness function will be the similarity function (see Section 3.1). The best-ranked chromosome (chromosome with the highest similarity value) of the entire population of a generation will be the rhythm inserted into the connection.

The genetic algorithm starts by applying a population kick-start process, which takes into consideration the starting node's rhythm. Using the rhythm modification function (previously described in the hill climbing section) on the starter rhythm, it is possible to create a population of X individuals by simply applying this function X number of times. This allows us to create a population pool of various multiple individuals, while still keeping some characteristics of our initial rhythm.

For the breeding selection method we chose to use a simple roulette selection, giving the best ranked individuals a higher chance to breed. The crossover method is a one point crossover between both chromosomes, however there is always a small chance of a new chromosome suffering a mutation. The mutation function is set so that a random active note is changed to an inactive note or vice-versa.

4.3 Discussion

We have made some experiments with these optimization processes after exploring a variety of parameters.

We tried multiple variations of N for the rhythm modification function (which is used in all the optimization algorithms), we found that having a N value that was too high might cause the optimization functions to converge too rapidly. So to provide a smooth progression we tested with values around the interval of 3 to 7 note modifications, the best and smoothest progressions we obtained was with a N value of 5.

Specifically in the case of the stochastic hill climbing optimization, the challenge was to find a T value "sweet spot" so to say, one that did not overcompensate worse solutions but also did not completely shut them down. Through various tweaking and experimentation we found that a value between 0 and 0.1 were garnering the best results. We settled on the value of $T = 0.07$ as it provided a probability of acceptance of about 20% on lower similarity rhythms.

For the genetic algorithm, we settled on multiple varied population sizes going from 100 to 2000 individuals, though the higher population sizes tend to slow down the algorithm significantly. We also applied elitism on top of the roulette selection (such as keeping 10% of the previous best individuals of a population), which had the side effect of creating rhythmic repetition. We also tested various mutation percentages, however the impact on the final rhythms and the generative process was very minimal, since it would only affect a single note.

Due to the progressive nature of the hill climbing method, connections that are generated using this optimization algorithm shows a fast and straightforward convergence, with usually shorter connection lengths. Contrarily to the genetic algorithm that usually shows a high volume of rhythms within its connections especially if using elitism, which creates a lot of rhythmic repetition. The stochastic hill climbing's connection lengths varies heavily, there were instances with both very short and others very long connection lengths, making this optimization process very varied in terms of rhythm sequence length.

In terms of processing time, the genetic algorithm is by far the slowest generative process, while the normal hill climbing is much faster. The stochastic hill climbing is very varied but by far quicker to generate then its genetic algorithm counterpart.

4.4 Optimization Filters

Filters are methods for shortening the sequences of rhythms resulting from optimizations, allowing the user more control over the generated solutions. Filters can eliminate specific unwanted note-maps according to a set of pre-defined rules. There are three types of filters that we developed:

- Point-to-Point Filter
- The Rank Filter
- The Repetition Filter

The **Point-to-Point Filter** takes into consideration the position of a rhythm within the connection and cuts all the rhythms whose positions are between position X and position Y.

The **Rank Filter** allows the elimination rhythms whose similarity value intersect with a specific value interval. When the progression is not monotonic, as it is the case of the stochastic hill-climbing, it may vary significantly from the original sequence of rhythms.

The **Repetition Filter** like the name entails, permits the elimination of rhythms whose similarity values are equal and appear consecutively in the rhythmic sequence. The option of defining a leeway is also available, such as keeping X number of repeated rhythms, for example keeping only two consecutive rhythms and eliminating the rest of the repeated rhythms.

5 Composition and Traveling the Graph

Percussion rhythmic composition can be viewed as a walk along Progressive Percussion Graphs. The variety of ways of traveling the graph offers the user multiple compositional options.

Regarding the metaphor of walking for the percussion composition process, one of the first things that had come to our minds was the idea of a wandering percussionist who would travel on a percussion graph from node to node, playing their respective connection along the way. It could be a "flaneur" that would start in a particular node and wander around without any particular goal; alternatively he could start in a particular node of the percussion graph and wander randomly until he found a goal node. In both cases the composition would be the sequences of percussion rhythms corresponding to the connections along the path. Note that in the first case, the musical promenade could stop either because the "flaneur" found himself in a dead-end node (without any outgoing connections) or because the number of visited nodes reached the limit. In the second case, besides the same two stop conditions, the composition also ends when the goal node is found. The resulting composition could include repetitions, i.e. the same connection more than once, when the graph structure allowed cycles.

Rhythm composition may also be viewed as a solution to a graph path finding problem where we start with a particular node and try to find a goal node. The final composition will be the musical connection path found. At its core, a path finding method searches a graph by starting at one point and exploring adjacent nodes until the destination node is reached, with the intent of finding the shortest route or just a route. For that we may use, as a toolbox, any of the available search algorithms, informed or non-informed and the percussion piece will be the path found and not all the explored connections used during the search process.

The different path-finding algorithms [15] have different properties and will produce a rich variety of compositions. For example, breadth-first search will produce the musical composition with the less number of connections and uniform-cost search will output the optimal composition in terms of the number of percussion rhythms. Depth-first search will produce longer pieces in general. Cycles will correspond to musical repetitions and the human composer may be interested in repeating rhythmic sub-pieces, setting-up a limit on the number of cycles in a piece. Using iterative deepening depth-first search will result in a repeating piece that expands dependent on the increase of the depth-limit in each algorithm iteration. Search can be polarized towards the goal node, using an heuristic search algorithm like best-first where the similarity metric is used as the heuristic function.

The human composer may wish that a musical piece be the shortest possible tour that visits all the nodes exactly once (it will be a solution of the asymmetric Traveling Salesman Problem). The circuit found will be a percussion composition and corresponds to the sequence of musical connections linking the consecutive circuit nodes. In case the percussion graph is not complete (i.e., a connection exist between each pair of vertices) the graph will be automatically completed: connections will be created for every pair of nodes using one of the optimizers. He may wish to visit some node before others and will try to find circuits that satisfy the defined constraints. To search for the optimized circuit we can use a simple local search algorithm like 2-OPT [4] to improve a nearest neighbor tour as an initial solution.

6 Future Work

Although our application did suffer through a lot of internal testing, it would be quite interesting to test it with multiple outside users, especially musicians. As of this moment one of the main focuses is to insure that the system is robust enough, so that data and honest feedback can be gathered from a wide range of users. These tests will be on the systems musicality and the interface.

We intend to extend our system by including multiple note velocities instead of limiting it to just two velocity types (active and inactive). Instead of having a constant tempo for an entire percussion graph, we intend to allow rhythms with different tempos in the same percussion. Its important to note that this would imply a different similarity metric, rhythm representation and alter the optimization processes (include note velocity and rhythm tempo).

References

1. Biles, J., Genjam: A genetic algorithm for generating jazz solos. In: Proceedings of the International Computer Music Conference, pages 131 (1994)
2. Brown, A.R., Wooller, R.W., Kate, T.: The Morph Table: A collaborative interface for musical interaction. Australiasian Computer Music Association (2007)
3. Burton, A.R., Vladimirova, T.R.: Generation of musical sequences with genetic techniques. Computer Music Journal 23(4), 59–73 (1999)
4. Croes, G.A.: A method for solving traveling-salesman problems. In: Operations Research, pp. 791–812 (1958)
5. Dahlstedt, P.: Autonomous evolution of complete piano pieces and performances. In: Proceedings of Music AL Workshop (2007)
6. Hoover, A., Stanley, K.: Exploiting functional relationships in musical composition. Connection Science (2), 227–251 (2009)
7. Horner, A., Goldberg, D.: Genetic algorithms and computer assisted music composition. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 437–441 (1991)
8. Horowitz, D.: Generating rhythms with genetic algorithms. In: Proceedings Of The National Conference On Artificial Intelligence, pp. 1459–1459. John Wiley & Sons Ltd. (1995)
9. Johanson, B., Poli, R.: GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters. In: Proceedings of the Third Annual Conference: Genetic Programming, pp. 181–186 (1998)
10. Koza, J.R.: On the programming of computers by means of natural selection, vol. 1. MIT Press (1996)
11. Miranda, E.R., Biles, J.A.: Evolutionary Computer Music, vol. (7). Springer (2007)
12. Monkey machine - the online drum machine, `http://rinki.net/pekka/monkey/`
13. Percussion Walker - A Percussion Graph Prototype, `http://bookmark.labmag.di.fc.ul.pt/?page_id=1966`
14. Phon-Amnuaisuk, S., Tuson, A., and Wiggins, G. Evolving musical harmonisation. In International Conference on Adaptive and Natural Computing Algorithms, pp. 1–9 (1999)
15. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall (2003)
16. Smith, Joshua R.: Designing Biomorphs with an Interactive Genetic Algorithm. In Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 535–538. Morgan Kaufmann Publisher (1991)
17. Tokui, N., Iba, H.: Music composition with interactive evolutionary computation. In: Proceedings of the 3rd International Conference on Generative Art, vol. 17, pp. 215–226 (2000)
18. Wooller, R., Brown, A.R.: Investigating morphing algorithms for generative music. In: Third Iteration: Third International Conference on Generative Systems in the Electronic Arts, Melbourne, Australia (2005)
19. Wooller, R., Brown, A.R.: Note sequence morphing algorithms for performance of electronic dance music. Digital Creativity 22(1), 13–25 (2011)
20. Yee-King, M.J.: The evolving drum machine. In: Music-AL Workshop, ECAL Conference (2007)