AUTOMATIC NOTATION OF COMPLEX RHYTHMS USING SIEVES IN DISSCO

Haorong Sun and Sever Tipei

Computer Music Project of the School of Music
National Center for Supercomputing Applications
University of Illinois

ABSTRACT

DISSCO, a Digital Instrument for Sound Synthesis and Composition, combines computer-assisted (algorithmic) composition and sound synthesis in a seamless, unified approach. Its output can take the form of electro-acoustic sounds, a printed score or both. In DISSCO, precise, unequivocal choices coexist with random options that could be introduced at all levels of a musical work. When transcribing the output of a computer-assisted composition into a musical score that uses traditional Western notation, the automatic notation of complex rhythms is arguably the greatest challenge. In DISSCO, this challenge is managed through the use of sieves, logical filters involving modulo and Boolean operations. Sieves restrict the number of available choices and guide their selection but, due to the randomness included in the process, a series of further adjustments are necessary in order to insure that the output is compatible with the tenets of traditional Western notation system. A number of methods that resolve these issues and achieve the desired result are described. The final output consists of a text file that follows the format of LilyPond, an open source software that can produce scores engraved according to traditional layout rules.

1. DISSCO PROJECT

A Digital Instrument for Sound Synthesis and Composition, DISSCO, combines computer-assisted (algorithmic) composition and sound synthesis in a seamless, unified approach. Its output can take the form of electro-acoustic sounds, a printed score or both.

In DISSCO, precise, unequivocal choices coexist with random options that could be introduced at all levels of both composition and sound makeup. This facilitates the production of multiple variants of the same work, creating a *manifold composition (1)*, when the random generator seed is modified. It also underlines the experimental attitude behind the project.

DISSCO is comprehensive in the sense that it does not permit the intervention of the user once the computations have started and it delivers a final product which does not require further adjustments. It is a “black box” meaning that it reads in the data provided by the user and outputs a finished object, the piece, in a uninterrupted process. This is necessary in order to preserve the integrity of the operation: modifying the results or intervening during the computations would amount to the alteration of the data or of the logic embedded in the software, a falsification of the experiment.

Recently, a concentrated effort has been directed toward the notation aspect, somewhat neglected in the past. Since randomness could be introduced during all stages of the process, challenges arise when trying to notate the output using traditional Western symbols. In DISSCO, such challenges are managed through the use of sieves, logical filters involving modulo and Boolean operations, first introduced by Iannis Xenakis (2).

1.1. Sieves

The most obvious usage of sieves is in creating pitch scales. However, rhythmic templates or selections operating on other aspects of a composition could be handled as well with the help of this powerful, abstract tool.

Constructing a sieve requires the presence of a grid of equally distanced elements such as the *equal temperament* tuning and the chromatic scale associated with traditional Western music notation. The desired set of pitches (eg. C major scale) is then selected from the total chromatic with he help of a sieve, as shown by Xenakis.

In the case of notated rhythms the situation becomes more complicated if along binary divisions of the beat. tuplets (divisions by three, five, etc.) are present. In order to accommodate all of them a grid is needed whose Elementary Displacement Unit or EDU is 1/LCM, (Least Common Multiple) of all selected divisions of the beat. If sixteens, triplets, quintuplets (1/4, 1/3 and 1/5) are present, the beat will have a value of 4 * 3 * 5 = 60 EDU and, for example, multiples of 15 or 60 / 4 will designate start times for all the sixteens in the piece. With the help of Boolean operations one could build more discriminating rhythms but a sieve of the form:

$$12_0 \cup 15_0 \cup 20_0 \quad (1)$$

will allow the use of quintuplets (12_0), sixteens (15_0), and triplets (20_0).

Weights can be attached either to individual moduli or to particular elements of the sieve output defining their probabilities of occurrence. This could be a valuable instrument for implementing a hierarchy between beats in a measure and between subdivisions of individual beats. In the following example, in a 3/4 meter, the down beat (0) receives the largest ponder followed by the third beat (8) while the second beat (4) is assigned a lower weight mimicking the distribution usually occurring in traditional music. Inside each beat, a similar arrangement is initiated:

sixteen	0	1	2	3	4	5	6	7	8	9	10	11
weight	20	5	12	8	10	2	5	3	15	5	10	8

table 1. Sieve weights

1.2. The problem.

In DISSCO, the automatic notation is handled by CMOD, the Composition Module. The time signature, the duration of the beat in EDUs (BeatEDUs), are selected by the user but the software generates in many cases random data. Accordingly, when designing the program, the following issues have to be considered:

1. sounds may be generated in a random sequence but the notation component needs to handle them according to the time order.
2. the start time and duration of a sound are randomly picked within a given range, but the end time could have a value incompatible with Western music symbols.
3. LilyPond (an open source software which can translates a particular text file into a .pdf score file) is used but the language syntax of LilyPond, was designed to be used by humans, not by computers.
4. Since different time signatures alone could change the entire score, it is necessary to design an algorithm that can manage various such situations.

2. THE DESIGN

2.1. Generating and adjusting sound data

Three classes belonging to CMOD, ***Note***, ***Matrix*** and ***Output*** participate in the notation process. ***Note*** holds all the features of a sound: start time, end time, pitch, loudness and "modifiers" or markings such as accents, pedal, 8va signs, etc. ***Matrix*** builds a 3D matrix (start time * duration * type of sound), insures that start times and durations are values compatible with the sieve and makes sure that sounds of the same type do not overlap. ***Output*** builds the text file .ly, which serves as input for LilyPond.

Due to the fact that the ***Matrix*** class may produce cases where the end time of sounds have values not allowed by the sieve and hence incompatible with the Western notation, it is necessary to:

1. Find the invalid-time remainder of the EndTime (EndTime mod BeatEDUs)
2. Construct a valid-time remainder vector inside the ***Matrix*** class (sieve values)
3. Use a binary search to locate the time spot closest to the invalid-time remainder and acquire the smallest absolute difference (difference = invalid_remainder - closest_valid)
4. Subtract the difference from the original EndTime
   (new_End = EndTime - difference)

The reason for choosing a binary algorithm in the second step instead of a linear search is that when notating scores that employ different kinds of tuplets, the size of the BeatEDUs is growing fast, along with the number of valid positions allowed by the sieve. Therefore, a linear search with time complexity O*(n)* will result in large running time. Since the valid positions are in a sorted array, applying a binary search with time complexity O*(log n)* will result in a much faster running speed.

For example, a linear search on a sorted array with size 32 will have, in the worst case, 32 operations. Applying a binary search on the same array will only result in 5 operations, also in the worst case. The advantage is even more obvious if the array size is doubled: 64

operations with the linear search, compares to 6 operations with the binary search.

2.2. Data structure

A 2-dimensional vector is constructed according to the time signature of the piece, every row representing a bar. Then the randomly generated sounds are sorted according to their start time and pushed into the vector.

The reason of building a 2D vector instead of a single dimension one containing all the sounds of the composition, is to divide a big problem into small ones. In traditional Western music, there is always an integer number of bars, each with a constant duration. By dividing the sounds into small groups, the restrictions and edge cases for the processing function can be narrowed down, which will be convenient for function design or debugging. Also, if the processing function does not need to consider complex situations, the function itself will have a higher time efficiency when executed.

After all the sound data of the 2D vector is entered, rests are added filling the gaps between sounds. Rests have the same object structure as notes, which helps in later calculations.

When a long sound or rest lasts across the bar line and is positioned across multiple rows of the 2D vector, it will be split into several parts, placed in the proper locations and marked as “continuous sound” so that it will be represented as a note with ties in the final generating process.

3. FINAL NOTATION

3.1 Main logic

If the value of BeatEDUs - the duration of one beat - is 60 and the time signature 4/4, then 60 will be a quarter note, 30 will be an 8th note and 20 will be an 8th note inside a triplet. However, using a different time signature, for example 6/8, the BeatEDUs (60) will count as an eighth note. In order to make the program compatible with different time signatures, the note type which represents a beat will be stored. Then, the notation function will select the other note types according to each value's proportion to the BeatEDUs.

Since the start time or the end time of a sound could be in the beginning, end, or middle of one beat, sounds and rests need to be split according to beats in order to be correctly notated. Every sound is processed individually and the method used to accomplish this involves three main steps:

1. If the sound starts in the middle of the beat it is necessary to split the sound first, in order to fill this beat, usually ending up with a tuplet.
2. If the remaining part of sound exceeds the BeatEDUs (60 in this case), it will be notated with relatively large notes values.
3. If the sound does not end at the end of a beat, its last part will become the beginning of the tuplet waiting to be filled in the next sound process.

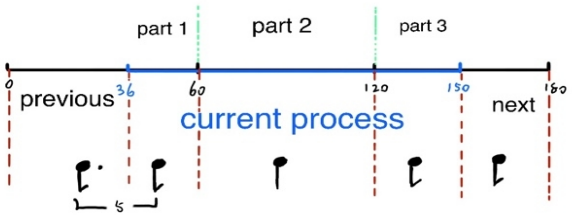

figure 1
Processing a sound that starts at 36 and ends at 150

3.2 Duration adjustments

Sometimes there are edge cases that deviate from the above logic. Sounds are created in random sequence and then sorted according to their start time (see 1.2). **Matrix** makes sure that valid sieve values are assigned to start times, durations and end times but rests - the time span between the end of a sound and the beginning of the next one - are not considered until later. Consequently, some rest durations may end up with values which are not allowed by the sieve and can not be notated.

In the example bellow which covers two beats (0-120) and shows two sounds surrounding a rest, BeatEDUs equals 60. The beginning and end times are expressed by the numbers between parentheses:

Sound1 (0,20) ***Rest*** (20, 45) ***Sound2*** (45, 120)

The sounds and the rest begin and end in valid places but while both sounds have

proper durations, the rest does not. The program will build an eight-note triplet for the first beat leaving a gap of size 40 to the end of the beat that the following rest and sound need to fill. However, the ***Rest*** has an invalid duration which can not be notated (45 - 20 = 25) .

The method we introduce to solve this problem is to check if the duration of the rest is consistent with the current tuplet type. If not, the rest and the next sound will be adjusted to complete the tuplet format.

This method involves 3 steps:

1. Find out the largest common divider between the duration of ***Sound1*** and BeatEDUs. Store the value as t_base
2. Find out a multiple of t_base that is close to duration of the rest
3. Adjust the duration and EndTime of the ***Rest*** as well as the StartTime of ***Sound2***.

The largest common divider of ***Sound1*** duration and BeatEDUs is 20, then the potential new duration for the ***Rest*** will be either 20 or 40, Since 20 is closer to the original duration, it will be chosen as the new duration. The original sounds and the rest will be changed as follows:

Sound1 (0,20) ***Rest*** (20, 40) ***Sound2*** (40, 120)

The remainder of the work is to output a text file following the LilyPond format (.ly) so that a music sheet can be generated accordingly.

4. CONCLUSION AND FURTHER WORK

When transcribing the output of a computer-assisted (algorithmic) composition into a musical score, the automatic notation of complex rhythms is arguably the greatest challenge. It can be facilitated by the use of sieves, abstract tools that offer a practical and rigorous solution. Sieves restrict the number of available choices and guide their selection but, due to the randomness included in the process, a series of adjustments are necessary in order to insure that the output is compatible with the tenets of traditional Western notation system.

Transcribing frequencies, basic dynamic markings as well as some other simple symbols (ped., 8va, accents, etc.) is a trivial task. Dealing with multiple lines and chords should also be a more or less straightforward exercise as LilyPond offers easy access to such features.

A main challenge for the future is to develop the capability of transcribing electro-acoustic events and combining them with vocal or instrumental parts in a coherent, general score. It is anticipated that LilyPond's capabilities will not be sufficient in managing this situation and that extra graphic devices will have to be included.

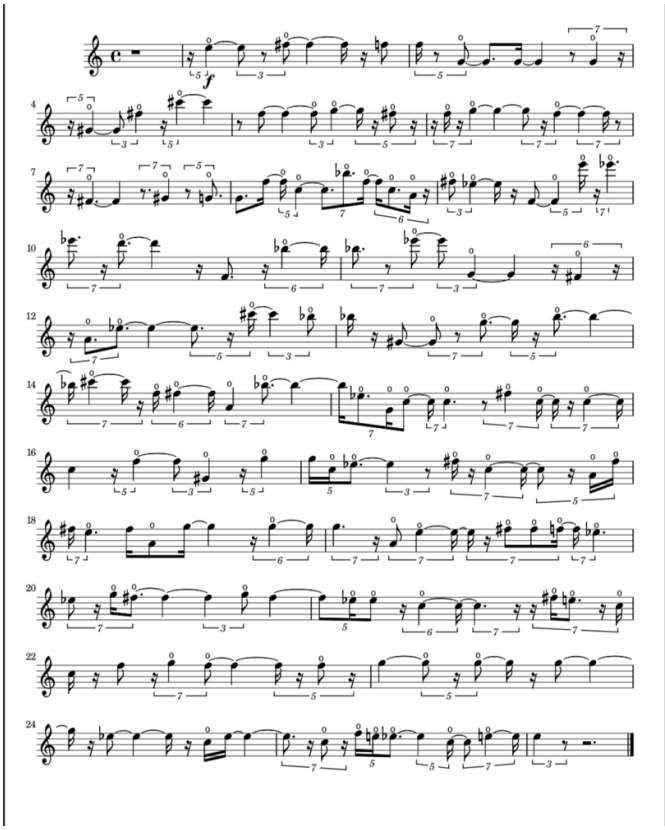

figure 2
Score printed automatically from random input

5. REFERENCES

1. Tipei, S. - *Manifold Compositions: Formal Control, Intuition, and the Case for Comprehensive Software*. Proc. 2007 Int'l Computer Music Conference (Copenhagen, Denmark, 2007), International Computer Music Association, San Francisco, CA, 2007, pp. 429-436

2. Xenakis, I. - *Formalized music,* Pendragon Press, 1992. p. 197.

Acknowledgment:
This work is supported by the NCSA SPIN program.