12-19-2008

Preferences in Musical Rhythms and Implementation of Analytical Results to Generate Rhythms

Shashidhar Sorakayala
University of New Orleans

Preferences in Musical Rhythms and Implementation of Analytical Results to Generate Rhythms

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
In partial fulfillment of the
Requirements for the degree of

Masters of Science
In
The Department of Computer Science

By

Shashidhar Sorakayala

B.Tech, Jawaharlal Nehru Technological University, India, 2003

December, 2008

Acknowledgements

I would like to express my gratitude to my professors, my parents, and my sister and to the number of people who became involved with this thesis, one way or another.

I am eminently grateful to my supervisor, Dr. N. Adlai A. DePano whose help, suggestions and encouragement helped me in all the time of research for and writing of this thesis. In addition, I would like to thank Dr. Shengru Tu and Dr. Golden G. Richard III for being on my thesis defense committee. It was a great honor having them both.

I am very much thankful to my family and friends for the continuous support and help they provided to me.

Table of Contents

List of Figures

ABSTRACT

Rhythm is at the heart of all music. It is the variation of the duration of sound over time. A rhythm has two components: one is the striking of an instrument – called the "onset" – and the other is silence. Historically, musical forms and works were preferred and became popular by their rhythmic properties. Therefore, to study rhythm is to study the underpinnings of all of music.

In this thesis, we explore basic rhythmic preferences in traditional music and, using this as a point of reference, methods are implemented to generate similar types of rhythms. Finally, a software platform to facilitate such an analysis is developed – it is the first of its kind available to our best knowledge as this research field has only recently emerged.

Keywords: The Evenness Method, Interval vector method, Euclidean method, Comparison measure, Erdos-deep method.

CHAPTER 1

INTRODUCTION

Rhythm is an important element of music.[5] It is the mixture of notes (onsets) and silences (rests). Composers of music can vary the timing of notes to make their composition more interesting and, hopefully, produce good music. With the invention of computers, generation of rhythms has become easier and more effective. The question arises: What type of rhythms should one generate? What are the properties that a rhythm should have in order to be effective? By studying various rhythms from the past and the reasons for their being preferred over others, many properties can be derived which then become tools in generating new rhythms.

The rhythms played in different parts of the world have their own specific properties. For example, it may be mandatory to play a note at a particular beat and only such rhythms are preferred in that particular region. Some popular rhythms in the past have all their notes well-spaced, *i.e*, all the notes are evenly distributed among rests. Some others may have a prescribed set of frequencies with which all the inter-onset durations are present. Our attempt here is to generate rhythms which exhibit such properties.

1.1 Representation of Rhythms[1, 5, 9]

In this thesis, we use three methods of representing rhythms. The first method is the binary representation. This consists of a vector of ones and zeros, where the ones represent notes and the zeros represent silence. For example, '1 0 1 0 0 0 1 0 0 1 0 0 1' represents a rhythm with five notes and eight silences. The number of time units in a rhythmic cycle is the number of notes plus the number of silences, which would be thirteen in the above example.

The second method is a geometric representation and makes use of a polygon inscribed in a circle. The polygon is used to efficiently represent the relative lengths of notes and silences on a circular lattice consisting of the circle's boundary divided into equal time units. Notes of the rhythm are represented by vertices placed at the appropriate lattice point which are then connected to form a polygon.

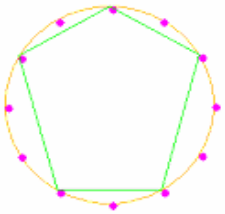
The third method is the histogram approach, which effectively visualizes the frequencies of all the interval lengths.[5] The interval length is the number of silences plus one between two consecutive notes in a rhythm. A histogram then represents the number of times each interval length is present in the given rhythm. The figure below illustrates each of these three methods of rhythm representation:

a. 1 0 1 0 0 1 0 1 0 0 1 0

binary sequence

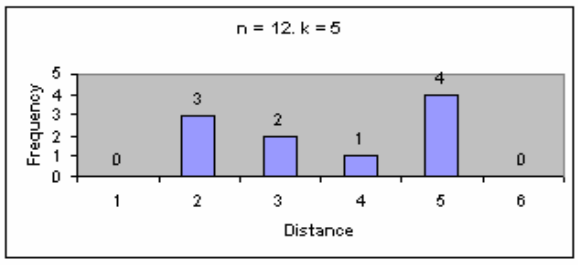
representation

b.



polygon representation

c.



histogram representation

Fig 1.1: Representation of Rhythms

1.2 Basic Concepts

Rhythm is a musical element that will remain for as long as music is made on this earth. Many rhythms were produced in the past but only certain ones have persisted and become popular in certain regions. By studying some of these popular rhythms, distinctive properties can be explored to help recognize and produce similar types of rhythms.

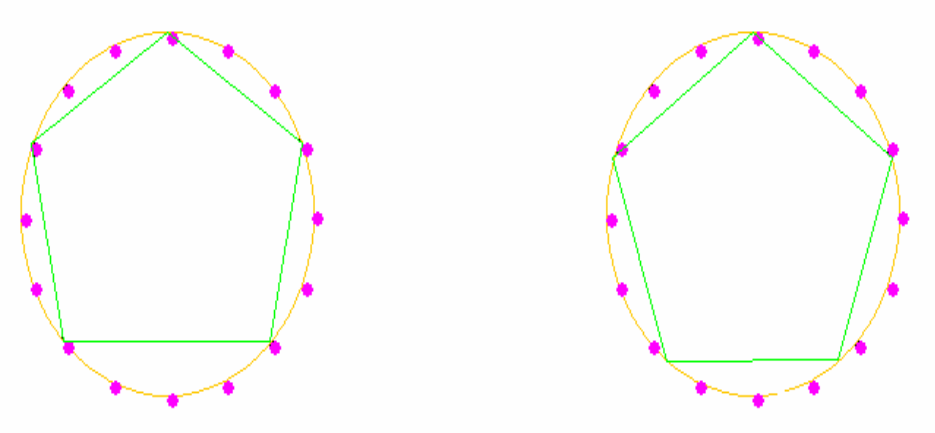
1.2.1 Evenness[1,4]

A rhythm that is even (or well-spaced) creates a sense of harmony in the minds of people and is usually generally liked. Many rhythms in the past have exhibited such properties of evenness to one degree or another. In fact, mathematical measures of evenness have been proposed that identify, if not explain, cultural preferences of rhythms in traditional music.[1] Douthet and Entringer have proposed a measure which simply adds all the interval arc-lengths (geodesics along the circle) determined by all pairs of intervals in a scale.[1] They proposed another measure which uses interval chord lengths as opposed to geodesics along the circle. However, it is possible to define a measure of rhythmic evenness that is not only more efficient to compute than these measures, but which is sensitive enough to discriminate well between rhythms. This is known as the Regression-evenness measure.[10]

For example, a rhythm called Bossa-Nova which first appeared in Brazil in the late 1950's, was a popular rhythm amongst the middle and upper class districts. It embodied free life style in Brazil and consists of five onsets (k=5) distributed uniformly among sixteen time units (n=16). Its binary sequence representation is:

1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0

This rhythm can also be represented by a polygon inscribed in a circle:



a. Bossa Nova rhythm **b.Optimal rhythm**

Fig 1.2.1: Polygonal representation

By observing Figure 1.2.1, one can see that the five notes of the Bossa Nova rhythm are almost distributed evenly – it is achieved by moving the optimal rhythm notes to the nearest lattice points in the time domain of sixteen units. If we added all the interval lengths of the Bossa Nova rhythm, the evenness value turns out to be 48.

1.2.1 Regression-evenness measure[9]

In Figure 1.2.1(b), one can observe that, if we inscribed a regular pentagon inside the circle, it is not possible to distribute five onsets evenly among the sixteen time units. The second, third, fourth and fifth notes are not at lattice points on the circle. The sum of these deviations may be taken to be a measure of the un-evenness in a rhythm. The regression value of the rhythm is 1.19. It is obtained by first dividing the number of time units with the number of onsets which is 3.2. This determines where the onsets should be placed to form a regular k-gon inscribed in a circle. Since the circular lattice is equally divided by the number of time units and since the onsets are placed only on the available positions, summing all the deviations of the onsets from their optimal positions gives the above regression evenness value.

1.2.1Maximum evenness[1, 4]

The above evenness measures bring up the question of which configurations of points (rhythms) achieve maximum evenness. Let us assume that we are given a circular lattice with *n* points (evenly spaced), and we would like to create a rhythm consisting of *k* onsets by choosing *k* of these *n* lattice points.

Therefore in order to achieve maximum evenness, a regular *k*-gon is constructed with one vertex coincident with one lattice point, and then move the remaining onset points to their nearest lattice points. In this way, rhythms with maximum evenness are generated.

1.3 Interval Vector [1, 4]

One may also examine the "spectrum" of the frequencies with which all the durations are present in a rhythm.[1] The spectrum of frequencies is called the interval vector. For example, take the rhythm represented by the binary sequence:

1 0 0 1 0 0 1

The frequency of the interval length one is 1 (occurring between the last onset and the first onset of the next rhythmic cycle) while the frequency of interval length two is 2 (occurring between the first and second onsets, and again between the second and last onsets). Therefore, the interval vector of the above rhythm is

[1, 2].[10] Let us take the rhythm known as Gahu which is popular amongst the people of southeastern Ghana. It was used for popular entertainment to celebrate life. The binary sequence of this rhythm is:

1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0

By observing the spectrum of frequencies, one can determine the interval vector which would be:

[0, 1, 2, 2, 1, 2, 1, 1]

The interval vector can also be represented graphically by a histogram bar chart (see Fig. 1.3 below).

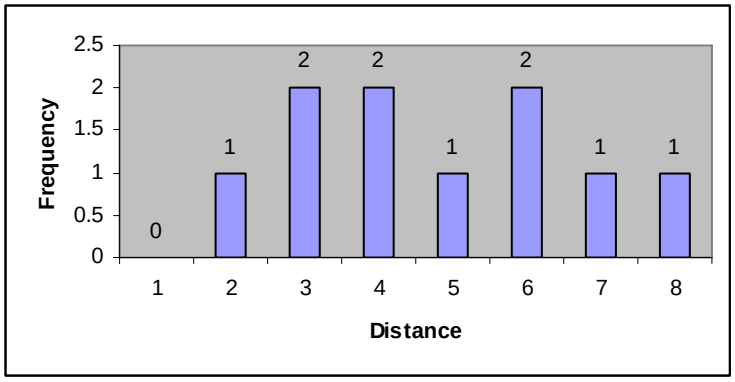


Fig 1.3: Gahu – Histogram representation

In the chart, one can observe that the frequencies are almost uniform, that is, each bar is either of height 1 or 2. Furthermore, the chart consists of a single connected component of occupied histogram cells, *i.e.*, all the consecutive interval lengths are covered. This observation suggests that the rhythms with a prescribed histogram shape as in Gahu may be preferable.[1] This geometric property could provide a heuristic for the discovery and automatic generation of other "good" rhythms. Therefore it would be desirable to be able to efficiently generate rhythms that either contain completely prescribed histogram shapes, or have geometric constraints on their shapes, or to find good approximations when such rhythms do not exist.

1.4 Erdos Deep property[1, 2, 4]

Another property that the preferred rhythms of the past have exhibited is the Erdos property.[2] In 1989, Paul Erdos asked whether one could find n points on the plane (no three on a line and no four on a circle) so that for every i, $i = 1, \ldots, n-1$ there is a distance determined by these points that occurs exactly i times. The cyclic rhythms with the Erdos property are referred to as deep rhythms. For example, consider again the rhythm Bossa-Nova

1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 0

which consists of five notes. If we focus just on the interval lengths that appear at least once between two notes, then we see that the frequencies of occurrence are unique. The interval vector for the rhythm is

[0, 0, 4, 1, 0, 3, 2]

which satisfies the Erdos-property.[1]

The analysis of cyclic rhythms suggests yet another variant of the question asked by Erdos. First, we note that if a rhythm $R[k, n]$ is such that $k \leq n/2$, then a solution to Erdos problem always exists: simply place points at positions 0, 1, 2,…, k. However, from a musicological point of view, it is not desirable to allow empty semicircles in the circumscribed polygon representation of the rhythm.

These constraints suggest the following problem: Is it possible to have k points on a circular lattice of n points such that for every i, $i = k(s)$, $k(s+1),\ldots, k(f)$, where s and f are pre-specified, there is a geodesic distance that occurs exactly i times, with the further restriction that there is no empty semicircle? It would seem desirable to be able to efficiently generate rhythms that satisfy the Erdos deep property with the additional constraints.

1.5 Euclidean Algorithm[3]

The well-known Euclidean algorithm computes the greatest common divisor of two given integers. The repeated subtraction until the remainder is one or zero in the Euclidean method actually generates repeated patterns which is a preferred property in rhythms.[3] Bjorklund faced a similar problem of generating repeated patterns but in a different context where for a given number of time intervals n, and a given number of signals $k < n$, the task at hand is to distribute the pulses as evenly as possible among those intervals.

This problem is similar to rhythm generation: given n time units, k ones (notes) and distributing these k ones evenly among $n - k$ zeroes.For example, if $n = 8$, and the number of ones $k = 3$, then the number of zeros $n - k = 5$. We outline the steps involved in the rhythm-generating algorithm:

Step 1.	[1][1][1][0][0][0][0][0]
Step 2.	1 1 1 0 0
	0 0 0
Step 3.	1 1 1
	0 0 0
	0 0

The rhythm thus generated is 1 0 0 1 0 0 1 0. A more detailed description of the procedure is as follows:

Step 1:First, place all the *k* ones and then place the $n - k$ zeros.

Step 2: Next, repeatedly take each zero and place it below ones equally. This continues until the arrangement consists of only one odd sequence. In this way we are actually distributing the ones uniformly among zeros. This is one way of generating rhythms where the onsets are uniformly distributed. Many rhythms from the past can be generated by using this method.

1.6 NoteWorthy Composer[18]

NoteWorthy Composer is a simple music composition software tool. It allows one to create, record, edit, print and play back one's own musical scores in pure music notation. One can also save the notation as a MIDI performance for use in other MIDI applications. Its print feature makes it possible to produce sheet music right from one's desktop. We find that this tool is sufficient for us to hear and notate the rhythms that we generate for different combinations of n and k.

Chapter 2

GENERATING EVEN RHYTHMS

Evenness is an important property in music. Music that is even has a soothing nature and is always preferred by listeners. A rhythm is said to be perfectly even when the onsets are distributed uniformly among silent nodes in the given time domain. We present here two techniques of generating rhythms that exhibit high evenness.

2.1 The Evenness Method

The first technique seeks to generate perfectly even rhythms. The procedure simply divides the number of time unit's *n* with the number of onsets *k*. The intermediate value *y* will be used to determine where the onsets should be placed to form a regular *k*-gon inscribed in a circle (which would be the polygon representation of a perfectly even rhythm).

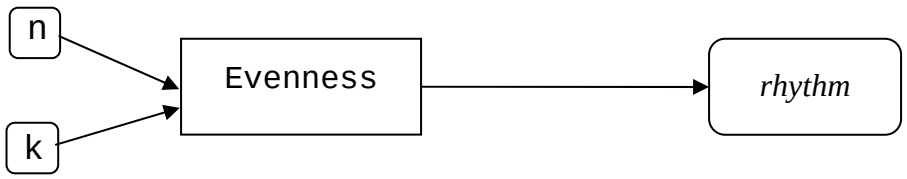


Fig 2.1: Evenness method

Procedure Evenness(n, k)

Input: n = number of equal time units

k = number of onsets

Output: Pcircle is the final *rhythm* generated. The onsets in this rhythm are uniformly distributed in the given time domain n. The rhythm is depicted as a binary number and also in *k*-gon format.

Auxiliary: a,y,z,temp = temporary variables

Pseudo code:

Step 1: Initializing the variables

- y = n/k
- z = 0.0
- temp = 0.0

Step 2:

WHILE z is less than n

- z = z + y
- The variable ‘a’ is set to the nearest integer less than or equal to z (a = (int)z)
- temp = z - a

IF temp is greater than 0.5 THEN

- The variable ‘a’ is set to the nearest integer greater than z (a = (int)z+1)
- Set the ‘a’th note in the rhythm Pcircle to 1

ENDIF; ENDWHILE

In the algorithm, we "construct" a regular *k*-gon with one vertex coincident with one lattice point (say at 0) and then move the remaining onset points to their nearest lattice points. Since the circle is divided into equal time units, the decimal is rounded off to the nearest integer and then the onset is placed at its nearest lattice point. In this way, the 1's are uniformly divided among 0's in the given time domain.

2.2 Euclidean method

This procedure is based on computing the greatest common divisor of two integers. This procedure generates large family of rhythms which were popular in traditional world music. These rhythms have a property that their onset patterns are distributed as evenly as possible.

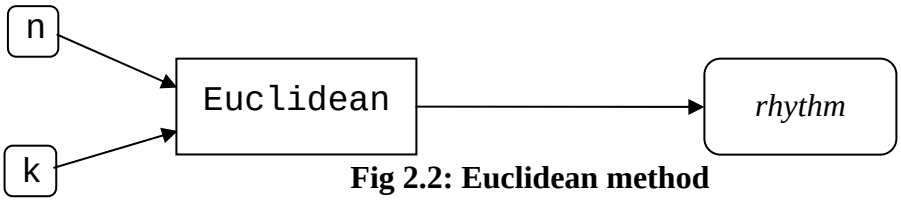


Fig 2.2: Euclidean method

Procedure Euclidean(n, k)

Input: n = number of time units

k = number of ones

Output: finalRhythm is the final rhythm generated. In this rhythm the onsets are uniformly distributed in the given time domain n. The rhythm is depicted as a binary number and also in *k*-gon format.

Auxiliary: rp_rhythm = repeating sequence in the rhythm

rm_rhythm = remainder column sequence

finalnum_rp = number of times the repeating sequence occurs in the rhythm

finalnum_rm = number of times the remainder column sequence occurs in the rhythm

size_rp = length of the repeating sequence

size_rm = length of the remainder column

dist1, dist2 = two possible distances between 2 onsets

i,j = temporary variables

Pseudo code:

Step 1: Initializing the variables

Initialization ()

finalnum_rp = k

finalnum_rm = n-k (number of silences)

rp_rhythm = 1

rm_rhythm = 0

size_rp = 1

size_rm = 1 EXIT

Step 2: Implementing Euclid algorithm

EUCLID (num_rp, num_rm)

Termination condition:

IF num_rm is 1 or num_rm is 0 THEN

finalnum_rp = num_rp

finalnum_rm = num_rm

ENDIF

ELSE

- IF num_rp is less than or equal to num_rm THEN
- num_rm = num_rm - num_rp
- num_rp remains the same
- Add the remainder column sequence to the repeating sequence
- Add size_rm to size_rp (size_rp = size_rp + size_rm)
- The remainder column sequence remains the same
- size_rm remains the same

ENDIF

IF num_rp is greater than num_rm THEN

- num_rp = num_rp - num_rm
- num_rm remains the same
- Add the remainder column sequence to the repeating sequence
- Add size_rm to size_rp (size_rp = size_rp + size_rm)
- The remainder column is equal to the previous repeating sequence
- size_rm is now equal to the previous size_rp

ENDIF

EUCLID (num_rp,num_rm)

EXIT (EUCLID)

Step 3: Printing the final rhythm

PrintRhythm ()

- Add the repeating sequence , finalnum_rp times to finalRhythm
- Add the remainder column sequence , finalnum_rm (0/1) times to finalRhythm
- Output finalRhythm

EXIT

The basic principle of Euclid's algorithm is to replace the larger of the two integers by their difference until both are equal. Bjorklund's algorithm is based on Euclid's principle except that the two integers are now replaced by the number of 1's and 0's. It uses the same repeated form of substraction just as Euclid did with his integers. This procedure actually distributes the onsets fairly uniformly in a given time domain.

2.3 Comparison Measure

The Regression Evenness Metric (REM) for a given rhythm measures how far the rhythm onsets are displaced from a perfectly even rhythm (with the same time frame and number of onsets) and is obtained by calculating the sum of all the onsets' deviations indicates the un-evenness in a rhythm. The rhythm which yields a minimum regression value is called a maximally-even rhythm.

Procedure REM(n, k,circle [n])

Input: n = number of time units

k = number of ones

circle [n] = rhythm in a binary format

Output: sumofdev - A value representing how far the given rhythm is deviating from the optimal rhythm.

Auxiliary: dist = inter onset distance in the optimal rhythm deviation is the deviation of an onset from its optimal position

temp, i, j, k = temporary variables

Pseudo code:

dist = n/k (optimal inter onset distance)

k = 0

FOR j = 0 to n DO

IF there is an onset at position j in the rhythm circle THEN

temp = k * dist

IF temp is greater than j THEN

deviation = temp - j

ENDIF

ELSE

deviation = j – temp

ENDIF

▪Add deviation to sumofdev

▪Increment k

ENDFOR

OUTPUT sumofdev

First divide the number of time units with the number of onsets. This determines where the onsets should be placed to form a regular k-gon inscribed in a circle. Finally sumofdev sums all the deviations of the onsets from their optimal positions on the circular lattice. The total deviation value indicates the unevenness in a rhythm.

Chapter 3

GENERATING FREQUENCY-BASED RHYTHMS

The rhythms of the past have exhibited a certain pattern in their frequencies of distances. Recall that the spectrum of frequencies is called the interval vector. A histogram is the best way to visualize the interval vector. After observing some of the traditional rhythms that have show enduring popularity, we find that the geometric shape histogram had an impact for their preference over others.

3.1 Interval vector method

Take for example, the rhythm called Gahu whose binary representation is:

1 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0

and whose frequency histogram is given by:

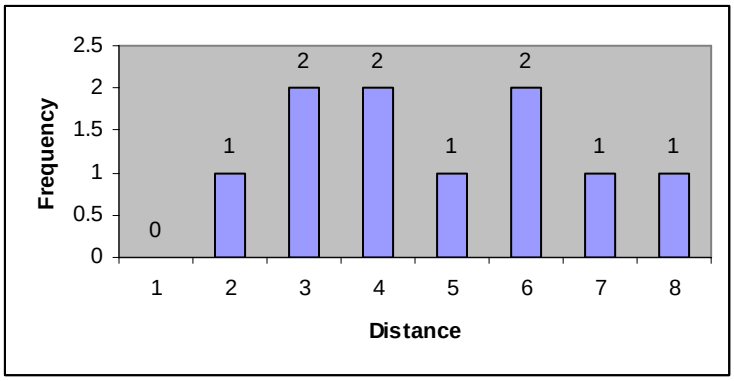


Fig 3.1: Gahu frequency histogram representation

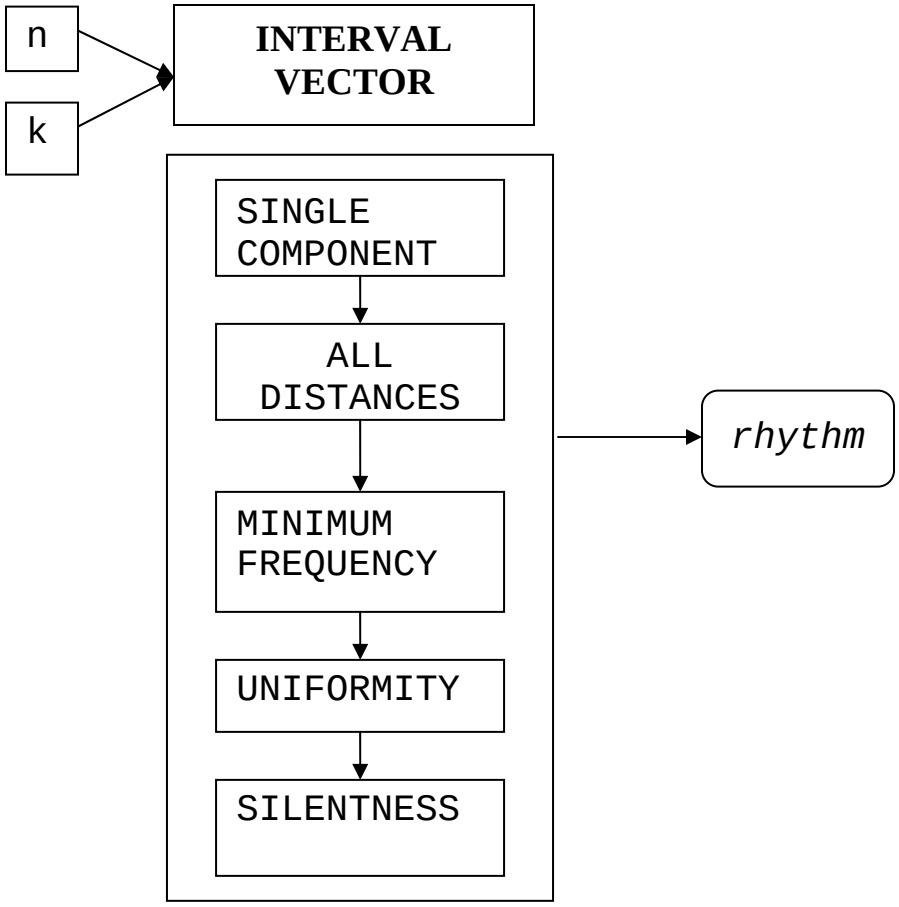


Fig 3.1.1: Interval vector method

Procedure Interval Vector(nm, km)

Input: nm = number of time units

km = number of ones

Output: circlefinal is the final rhythm generated and arrayfinal is the interval vector of the rhythm. A histogram visualizing the interval vector and a k-gon are also depicted.

gapfinal = the number of gaps in the histogram of the final optimal rhythm

zerofreq = the number of zerofrequency distances in the histogram of the final optimal rhythm

maxfinal = the maximum frequency in the histogram of the final optimal rhythm

difffinal = the difference between the maximum and the minimum frequency in the histogram of the final optimal rhythm

maxgapfinal = the maximum gap of all the gaps between consecutive onsets on the circle for the final optimal rhythm

Auxiliary = i, j, c1, c2, c3, c4, k2, d1, d2, finalcount, count, tmparr[1000], s,arraytemp[1000], gapresf, again, val, gap, numconv, maxfreq, sum, circlegap, midpnt, diff1, diff2, zerotemp, maxgaptemp = temporary variables

numgaps = number of gaps in the histogram obtained when an onset is placed in an optimal position

conversions = number of zerofrequency distances that get covered when an onset is placed in its optimal position.

flatsum = the sum of frequencies of those distances which get incremented on placing the onset in its optimal position.

gaponcircle = the gap on the circle (between onsets) in which the optimal position on the onset lies

midpoint1 = midpoint of the gap on the circle in which the position of the onset lies

maxcovdist = maximum distance covered on placing an onset in a particular onset

minfreq = minimum frequency in the histogram

freqdiff = difference between maximum and minimum frequency in the histogram

maxgap = maximum gap on the circle of all the gaps between consecutive onsets

maxcover = it is the maximum inter onset distance which is covered on placing an onset in a particular position

value = optimal position of the onset

mindist = shortest distance between two onsets on the circular lattice

Pseudo code: FOR finalcount = 1 to nm/2 DO

//initialization

- Place an onset at 0^{th} position of the rhythm 'circle' (circle[0] = 1)
- Place an onset at a distance of finaldist to the left of 0^{th} position of the rhythm (circle[nm - finalcount]=1)
- Increment the frequency of occurrence of the inter onset distance 'finaldist'(freqarray[finalcount]++)

//loop run for each onset placed

FOR i = 0 to km – 2 DO

- Initialize numgaps, maxfreq, flatsum, gaponcircle to a high value
- Initialize conversions to 0

//checking for the optimal position by placing the onset at various non covered instants of time

FOR j = 0 to nm DO

IF circle[j]==0 THEN

- Intialize count, gap, numconv, sum to 0
- Initialize value to j
- Initialize gapres to ‘n’
- Set arraytemp equal to freqarray

//Find the distances between the position of the onset and the other already placed onsets

FOR k2=0 to nm DO

IF circle[k2]==1 THEN

- d1 = mod(value – k2)
- d2 = nm - d1
- mindist = minimum (d1,d2)
- Increment count

//store the distances between the newly placed onset and other already placed onsets in a tmparr array.

- tmparr[count] = mindist

```
//increment numconv if a zero frequency distance is incremented on placing the onset.
IF arraytemp[mindist] == 0 THEN
Increment numconv
ENDIF
▪Increment arraytemp[mindist]
▪sum = sum + arraytemp[mindist]
ENDIF;ENDFOR

//Find the maximum inter onset distance which is covered on placing the onset in the jth position
maxcover=0
FOR c1 = 1 to count  DO
IF tmparr[c1] >  maxcover THEN
maxcover = value
ENDIF
ENDFOR

//Find the number of gaps formed in the histogram on placing the onset in the jth position.
gapres='n'
FOR c1 = 1 to nm/2 DO
IF arraytemp[c1] == 0  and gapres == 'n' THEN
▪Increment gap
▪gapres='y'
ENDIF
```

```
ELSE IF arraytemp[c1] == 1 THEN
gapres='n'
ENDIF
ENDFOR
IF arraytemp[nm/2] ==  0 THEN
 Decrement gap; ENDIF

//Find the maximum frequency in the histogram on placing the onset in the jth position.
maxm=0
FOR c2=1 to maxcover DO
IF arraytemp[c2] > maxm THEN
maxm=arraytemp[c2]
ENDIF; ENDFOR

//Find the gap on the circle in which the new position of the onset lies.
FOR c2=0 to nm DO
IF circle[c2] = 1 or c2 = n THEN
IF c2 > 'value' THEN circlegap = c2-c4;break
ENDIF
c4 = c2
ENDIF; ENDFOR
midpoint1=(c2+c4)/2
ENDIF;ENDFOR
```

Choose a position for placing the onset in such a way that

- gap is minimum
- numconv is the maximum.
- maxm is the least
- sum is the minimum
- circlegap is minimum
- The position should be nearest to the midpoint of the gap between onsets in which the position lies

// After placing the onset in the best possible position based on the above criteria , update the frequency of each possible inter onset distance in this rhythm

FOR c2=0 to nm DO

IF there is an onset in the c2th position in 'circle' rhythm THEN

- d1 = mod(value – c2)
- d2 = nm – d1
- Increment the minimum(d1,d2)

ENDIF

ENDFOR

ENDFOR

// After placing all the onsets in the best possible positions on the circular lattice, find the number of zero frequency distances in the histogram of the rhythm.

zerotemp=0;

FOR c3=1 to (nm/2) DO

IF freqarray[c3] is equal to 0

Increment zerotemp

ENDIF; ENDFOR

// Find the range i.e. difference between the maxium and the minimum frequency in the histogram.

- Minfreq = minimum(all elements of freqarray)
- Freqdiff = maxfreq - minfreq

```
// Find the maximum gap of all the gaps between consecutive onsets in the circle for the rhythm
generated
maxgap=0
maxgaptemp=0
gapresf='n'
 FOR c3=0 to n DO
IF circle[c3] == 1 or c3 == n THEN
IF maxgaptemp >= maxgap THEN
▪maxgap = maxgaptemp
▪maxgaptemp=0
ENDIF
ENDIF
ELSE IF circle[c3] is equal to 0
Increment  maxgaptemp
ENDIF
ENDFOR
```

At this stage , all the onsets are placed and a rhythm is formed

- Of all the rhythms so obtained in this loop with various initial distances, the final optimal rhythm is obtained based on the following criteria
- Numgaps is minimum (set numgaps to gapfinal)
- Zerofreq is minimum (set zerotemp to zerofreq)
- Maxfreq is minimum (set maxfreq to maxfinal)
- Freqdiff is least (set freqdiff to difffinal)
- Maxgap is minimum (set maxgap to maxgapfinal)

ENDFOR

Procedure explanation:

1. Place an onset at point 0 and another onset at a distance of one to it.

2. Next place the remaining onsets at one of the not covered points, such that it satisfies the following criteria.

- On placing the onset, the number of gaps in the histogram should be minimum. This will make ensure that we finally get a single connected component (number of gaps=0) or with minimum gaps.
- On placing the onset maximum number of distances should be covered. Since more zero frequency distances are covered, the maximum frequency will be minimum.
- On placing the onset, the maximum frequency should not increase .If it increases at all points, minimum is selected.
- In order to maintain relatively flat histograms the average of all the distances whose frequencies are increased on placing the onset should be minimum.
- In order to reduce silent ness, choose a position for the onset such that it is placed in the largest gap on the circle, thus reducing the silent ness.
- Another criterion is that the position of the onset is near to the midpoint on the largest gap in which it is placed. This will make sure that the silent ness is reduced to a maximum.

In this method at every stage an onset is placed, its position is chosen such that it satisfies the above conditions in the given order. Now the optimal rhythm may not cover distance one. Therefore this process is repeated by setting up various possible initial distances. The rhythms obtained are compared based on the following conditions in the given order and the best is chosen.

- The number of gaps in the histogram generated should be minimum. A histogram with a single connected component is more preferred.
- The number of zero frequency distances should be maximum .This will make sure that all possible distances are covered.
- The rhythm whose maximum frequency is less is preferred.
- The rhythm with a uniform and relatively flat histogram is more preferred .In order to maintain a flat histogram, the difference between the maximum and minimum frequency of the inter onset distances should be minimum.
- The rhythm with minimum silent ness is preferred.

3.2 Erdos-deep method

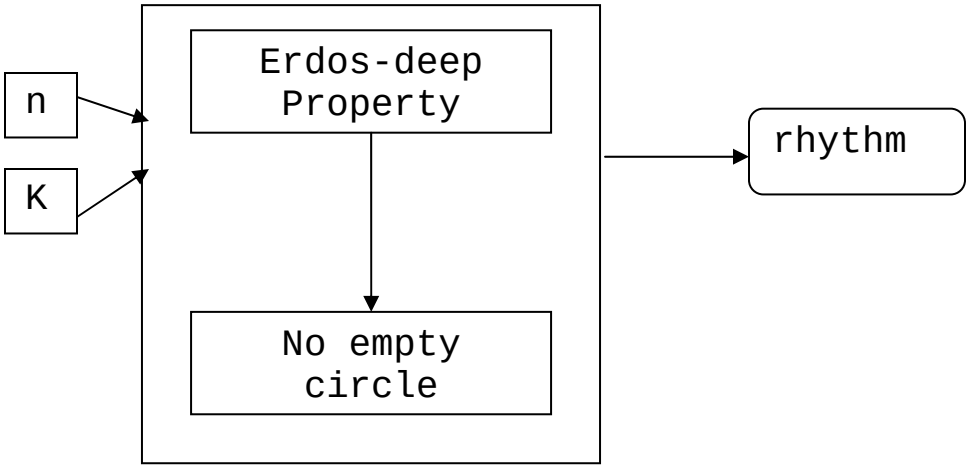


Fig 3.2: Erdos-deep method

Erdos-deep property[1]: We characterize which sets of k points chosen from n points spaced evenly around a circle have the property that, for each i = 1, 2,...,k−1, there is a nonzero distance along the circle that occurs as thedistance between exactly i pairs from the set of k points. Rhythms with this property are called Erdos-deep.

Procedure ErdosDeep_Pattern(nm, km)

Input :nm - number of time intervals

km – number of onsets

Output : patternPossible : Is true if an Erdos deep rhythm with no empty semicircles is possible with the given values of nm and km , else it is set to false.

Auxillary:

pattern - Rhythm with ‘nm’ time intervals

IsErdos - Boolean variable which is true if the rhythm is Erdos Deep

noEmptySemiCircle - Boolean variable which is true if the rhythm does not have a empty semi-circle

i,j,temp,pt,a[] – temporary variables

freqOfDistance – Gives the frequencies(number of times) of

inter-onset distances (from 1 to km-1) in the rhythm ‘pattern’

distance – inter onset distance

gapConsecOnsets – Gap between consecutive onsets on the circular lattice

Pseudo code:

- For a given combination of nm and km, initialize a ‘pattern’ for the rhythm.
- Initialize frequencies of all inter onset distances to 0.

```
For i = 0 to nm DO
IF pattern[i] = 1 THEN
distance = 0
FOR j = i + 1 to nm DO
IF  pattern[j] = 0  THEN
Increment distance
ENDIF
ELSE IF pattern[j] = 1 THEN
Increment distance
IF distance > (nm – distance) THEN
Increment freqOfDistance[distance]
ENDIF
ELSE IF distance >= (nm – distance) THEN
Increment freqOfDistance[nm - distance]
ENDIF
ENDIF
ENDFOR
ENDIF
ENDFOR
```

```
//Check if pattern is Erdos Deep (Is_ErdosDeep)
FOR i = 1 to km -1 DO

// Check if there is a geodesic distance j with frequency i ,  in the rhythm ‘pattern’ (
ExistsDistance)
FOR j = 1 to nm/2 DO
IF freqOfDistance[j] = i THEN
The rhythm ‘pattern’ is Erdos Deep
ENDIF
ENDFOR
ENDFOR

// Check if the pattern is a empty semi circle (NoEmpty_SemiCircle)
gapCosecOnsets = 0
FOR j = 0 to nm DO
IF pattern[j] = 1 THEN
 // j is the position of the first onset in pattern
break
ENDIF
ENDFOR
```

```
FOR i = j + 1 to nm + j DO
IF i is greater than or equal to nm THEN
pt = i - nm
ENDIF
ELSEIF
pt = i
ENDIF
IF pattern[pt] = 0 THEN
Increment gapConsecOnsets
ENDIF
ELSE IF pattern[pt] = 1 THEN
Increment gapConsecOnsets
IF nm % 2 = 0 THEN
temp = nm/2
ENDIF
ELSEIF
temp = (nm + 1) / 2
IF gapConsecOnsets >= temp THEN
There is an empty semi circle in the rhythm
ENDIF;ENDIF
ENDIF;ENDFOR
```

There is no empty semi circle in the rhythm

▪If the rhythm ‘pattern’ is Erdos Deep and there is no empty semi circle , then patternPossible is set to true

▪Else the above process is repeated for another permutation of the rhythm ‘pattern’.

```
//Code for permutation of the rhythm to get another rhythm
FOR i = nm – 2 to  0 DO
 IF a[i + 1] > a[i] THEN
FOR  j = nm – 1 , a[j] is less than a[i] DO
Swap a[i] and a[j]
FOR j = 1 to (nm-i)/2 DO
Swap a[i+j] and a[nm-j]
ENDFOR
FOR j = 0 to nm DO
pattern[j] = a[j]
ENDFOR
ENDFOR
ENDIF
ENDFOR
```

Given the number of time units (n) and number of onsets (k), $n_{c\ k}$ combinations of rhythms are generated. Then each rhythm is verified whether it satisfies Erdos-deep property and no empty-circle constraints. If any one rhythm satisfies the above constraints, it proves that for a given input, their exists a rhythm which satisfies Erdos-deep property with no empty circle. The rhythm generated is depicted as a histogram.

The function Is_ErdosDeep ( ) verifies whether the rhythm satisfies Erdos-deep property. If it satisfies then the function NoEmpty_SemiCircle( ) checks whether the rhythm has an no empty semi-circle. If both the functions return true, then the rhythm is said to satisfy Erdos-deep property with no empty circle.

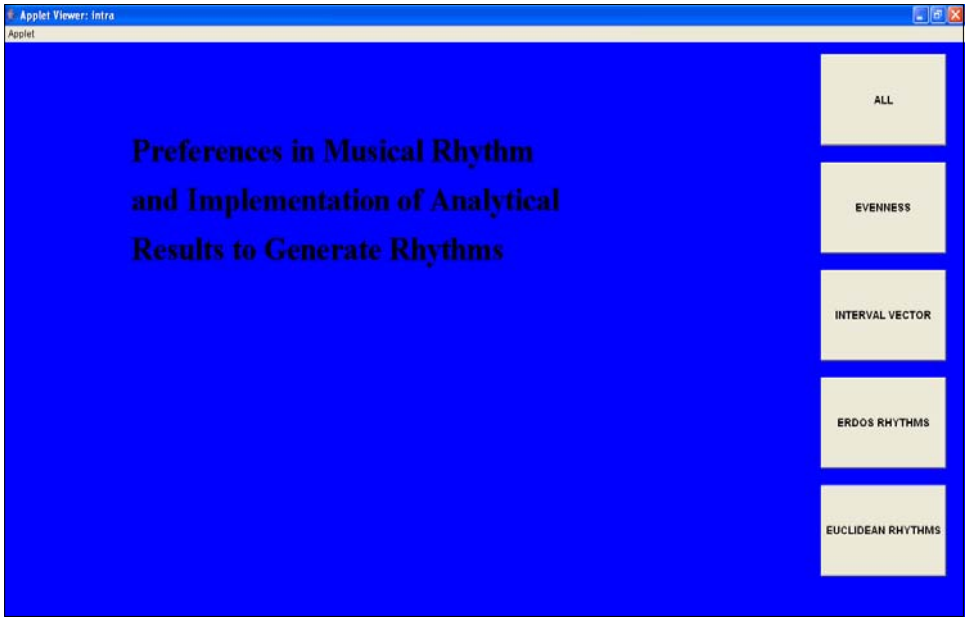
Chapter 4

SAMPLES OF GENERATED RHYTHMS

The software that was developed was supplied with an interface using Java applets. The generated rhythm was then represented in traditional musical notation using an application program called Noteworthy Composer.

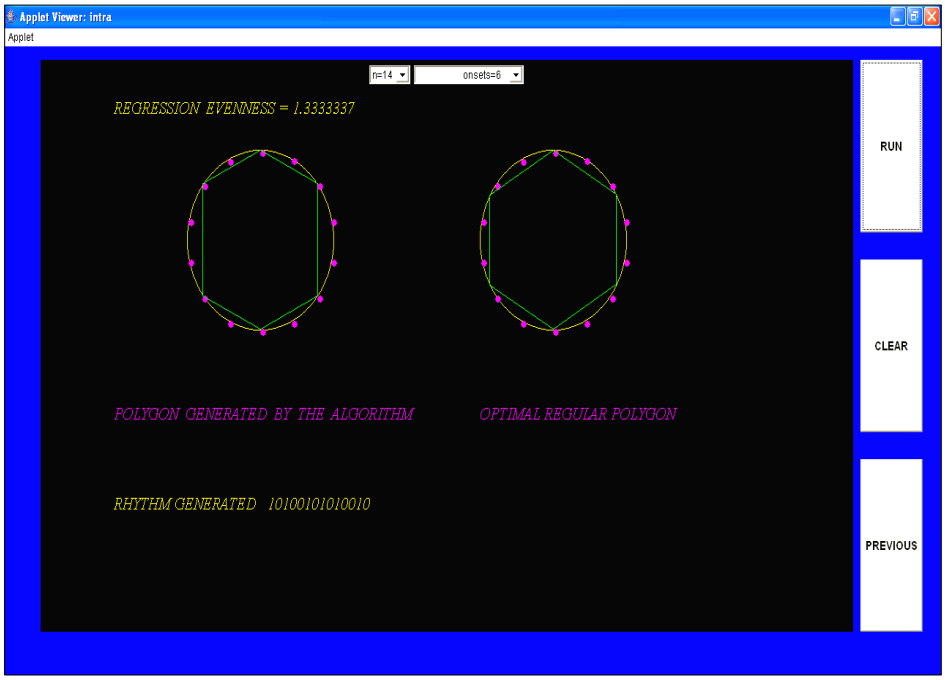
The initial screen of the interface has five buttons which are named **ALL**, **EVENNESS**, **INTERVAL VECTOR**, **ERDOS RHYTHMS** and **EUCLIDEAN RHYTHMS**. We illustrate this below:

Initial screen:



EVENNESS − Hitting this button, launches a new screen that displays rhythms generated using the 'evenness' procedure. This new screen has three buttons itself: **RUN**, **CLEAR**, and **PREVIOUS**. It also has two dropdown menus for selecting the value of *n* (number of time units) and *k*, the number of onsets. We illustrate this below:

Evenness screen:



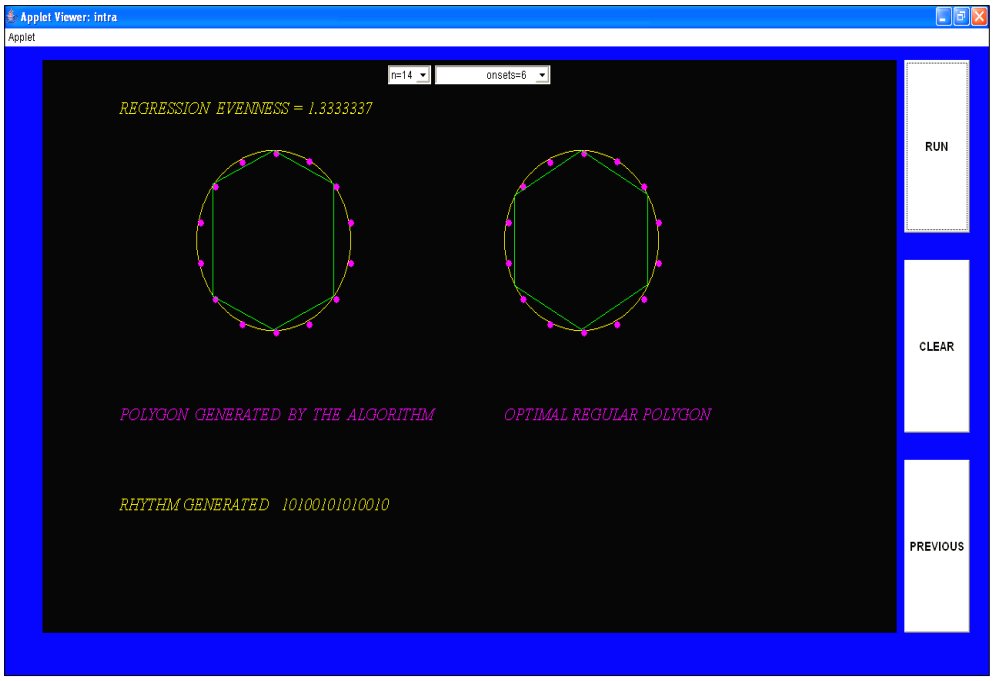
RUNAfter selecting *n* and *k* from the dropdown menu, hitting the **RUN** button generates a rhythm using the 'evenness' procedure. The output is depicted in binary format and also in a *k*-gon format. For comparison purposes, an optimal rhythm is also depicted in a *k*-gon format for the same combination of time units and onsets.

CLEAR This button clears the screen.

PREVIOUSThis button takes you to the previous screen.

INTERVAL VECTOR – Hitting this button launches a new screen that displays rhythms generated using the 'interval vector' procedure. This new screen has three buttons like the previous technique: **RUN**, **CLEAR**, and **PREVIOUS**. Likewise, it has the two dropdown menus for selecting *n*, the number of time units, and *k*, the number of onsets. We illustrate this below

Interval Vector screen:



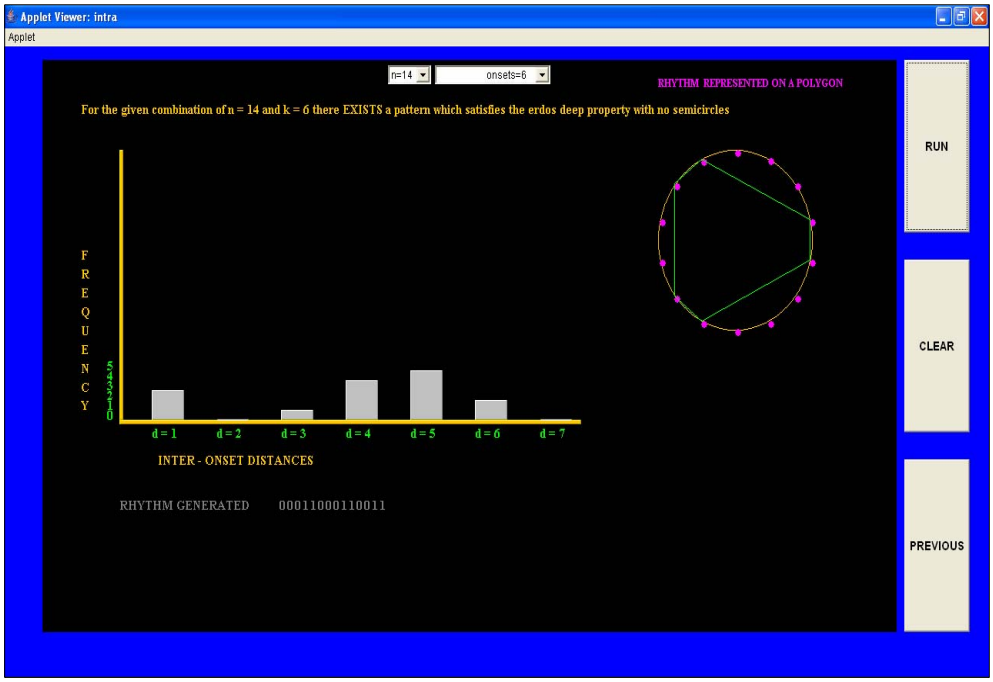
RUNAfter selecting *n* and *k* from the dropdown menu, hitting the **RUN** button generates a rhythm using the 'interval vector' procedure. The rhythm is depicted in binary format, and also in *k*-gon format. Additionally, the output's histogram is displayed. A sample output is given below:

CLEARThis button clears the screen.

PREVIOUSThis button takes you to the previous screen.

ERDOS RHYTHM – Hitting this button launches a new screen that displays rhythms generated using the 'Erdos pattern' procedure. As with the preceding techniques, this new screen has three buttons itself: **RUN**, **CLEAR**, and **PREVIOUS**. The two dropdown menus for selecting *n*, the number of time units, and *k*, the number of onsets, are present as well.

Erdos Rhythm screen:



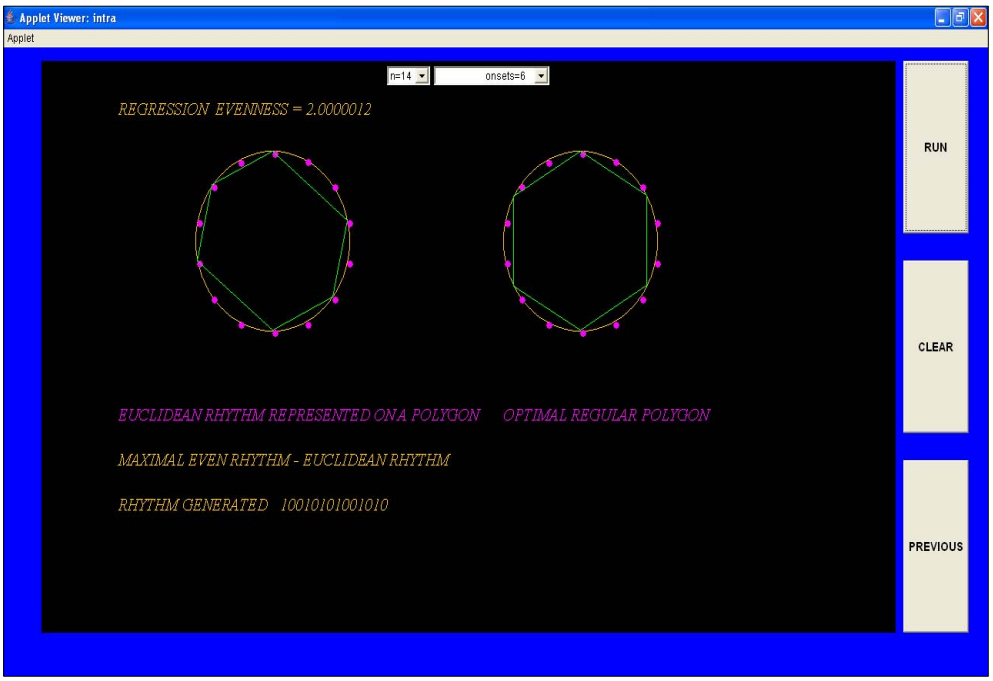
RUNAfter selecting *n* and *k* from the dropdown menu, hitting the **RUN** button generates a rhythm using the 'erdos_pattern' procedure. As in the preceding methods, the rhythm is depicted in binary format, in *k*-gon format, and with a histogram.

CLEAR This button clears the screen.

PREVIOUSThis button takes you to the previous screen.

EUCLIDEAN RHYTHMS – Hitting this button opens up a new screen which displays rhythms generated using the 'Euclidean' procedure. The screen has the now-familiar three buttons – **RUN**, **CLEAR**, and **PREVIOUS**. It also has two dropdown menus to select *n*, the number of time units and *k*, number of onsets.

Euclidean Rhythm screen:

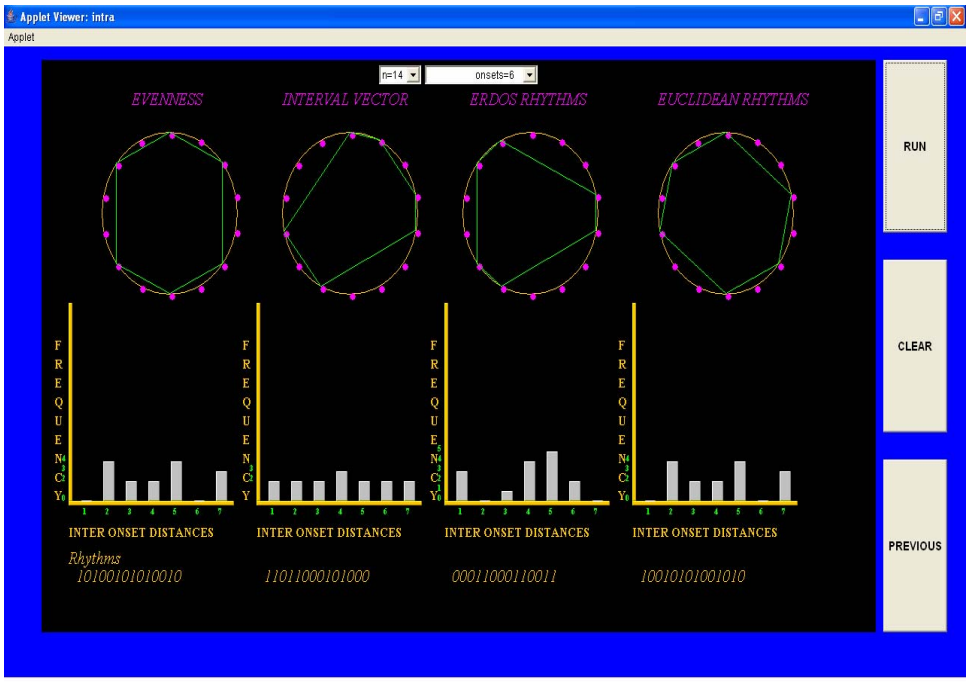


RUNAfter selecting *n* and *k* from the dropdown menu, hitting the **RUN** button generates a rhythm using the 'euclidean' procedure. As in the preceding methods, the rhythm is depicted in binary format, in *k*-gon format, and with a histogram.

CLEAR This button clears the screen.

PREVIOUSThis button takes you to the previous screen.

ALL – Hitting this button, opens up a new screen which displays rhythms using all the four procedures (Evenness, Interval vector, Erdos_pattern, Euclidean). The rhythms are depicted as a binary number, *k*-gon and as a histogram.



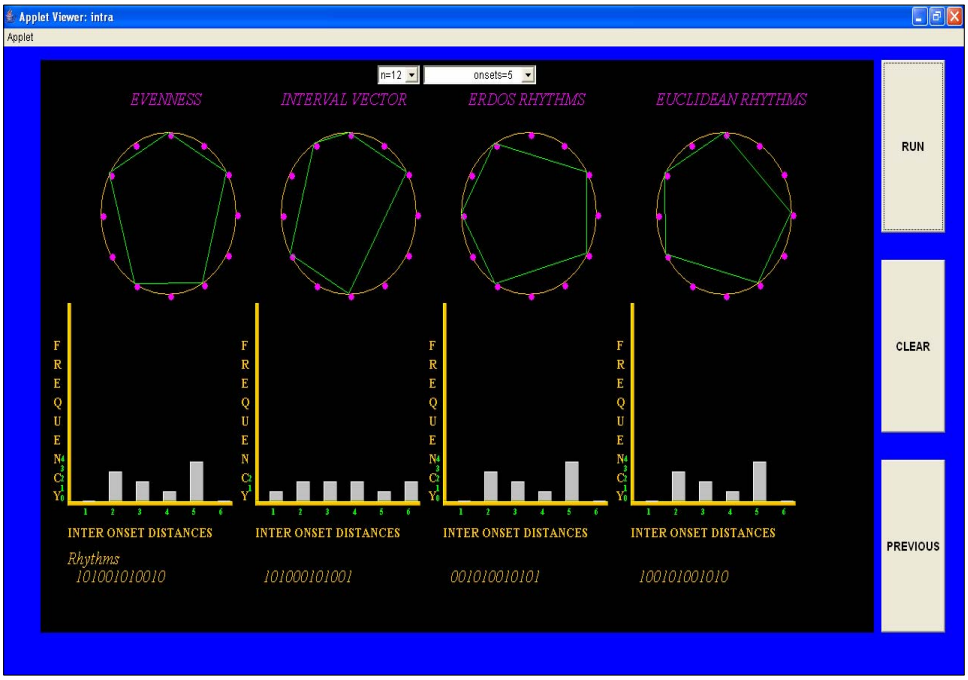
RUNBy selecting *n* and *k* from the dropdown menus, hitting this button generates rhythms using all four procedures (Evenness, Interval vector, Erdos_pattern, Euclidean). The rhythms are depicted in binary format, in *k*-gon format, and with a histogram.

CLEAR This button clears the screen.

PREVIOUSThis button takes you to the previous screen.

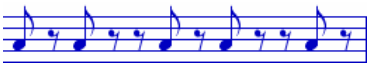
The screens below illustrate rhythms generated for different combinations of *n* (number of time units) and *k* (number of onsets). [**Note**: These rhythms are available in MP3 format and can be played from the accompanying CD.]

1. $n = 12, k = 5$

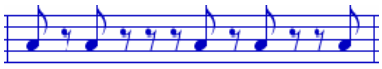


These rhythms are illustrated here in regular music notation:

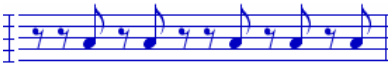
a. **Evenness**



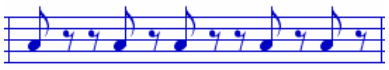
b. **Interval vector**



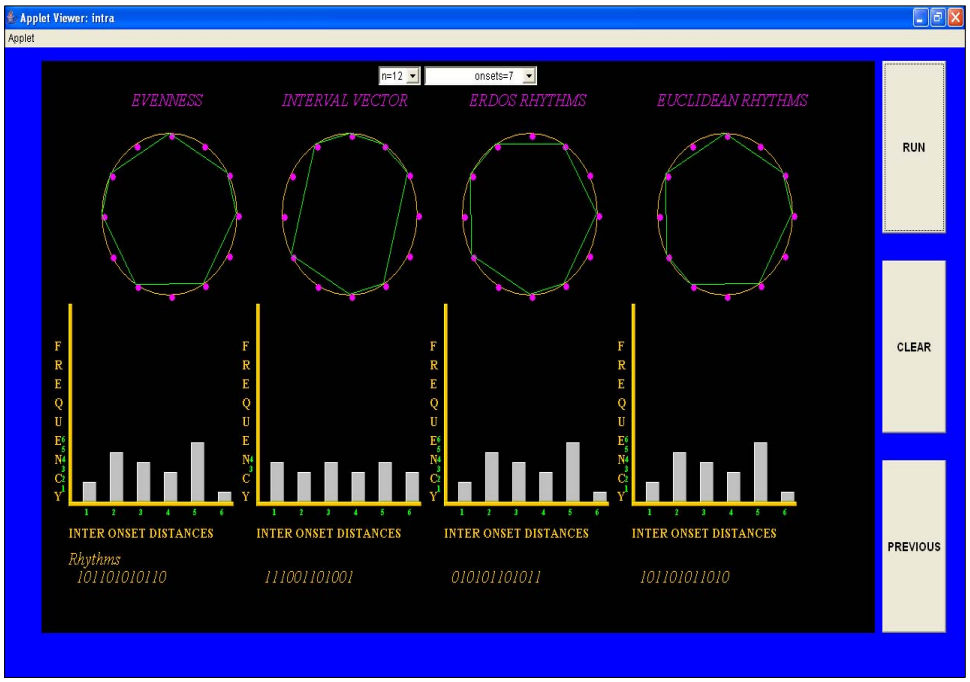
c. **Erdos rhythms**



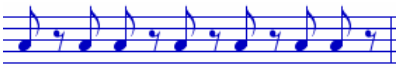
d. **Euclidean rhythms**



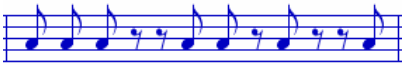
2. $n = 12$, $k = 7$



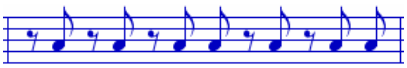
a. Evenness



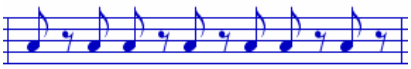
b. Interval vector



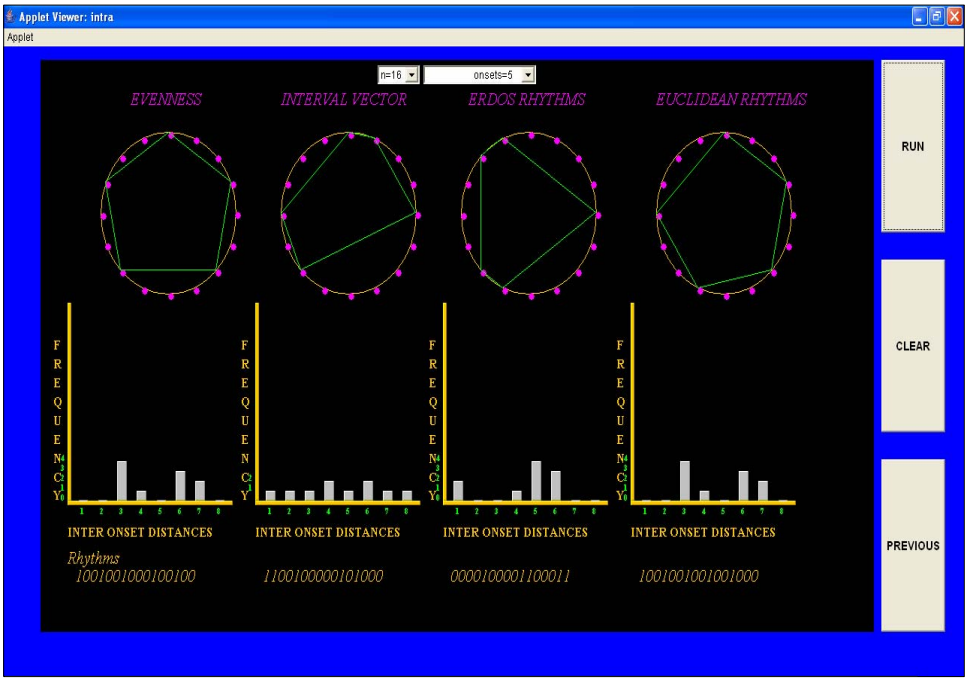
c. Erdos rhythms



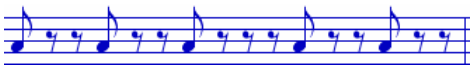
d. Euclidean rhythms



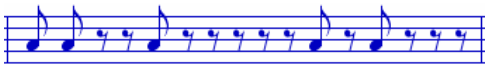
3. $n = 16$, $k = 5$



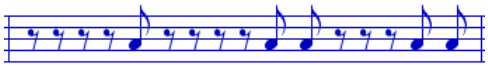
a. Evenness



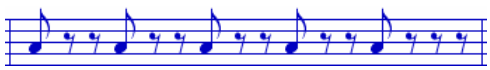
b. Interval vector



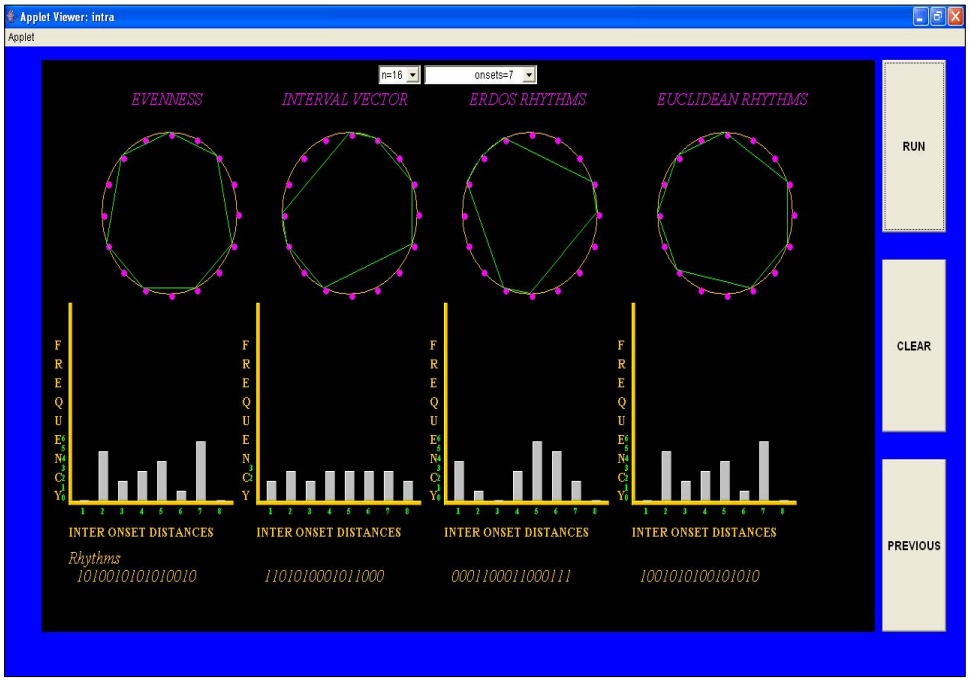
c. Erdos rhythms



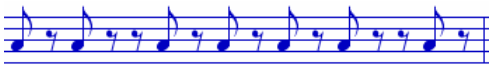
d. Euclidean rhythms



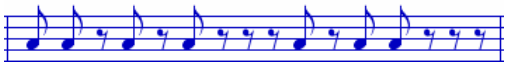
4. $n = 16, k = 7$



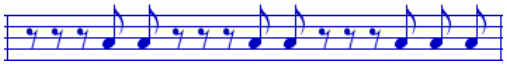
a. Evenness



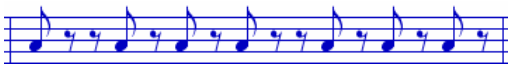
b. Interval vector



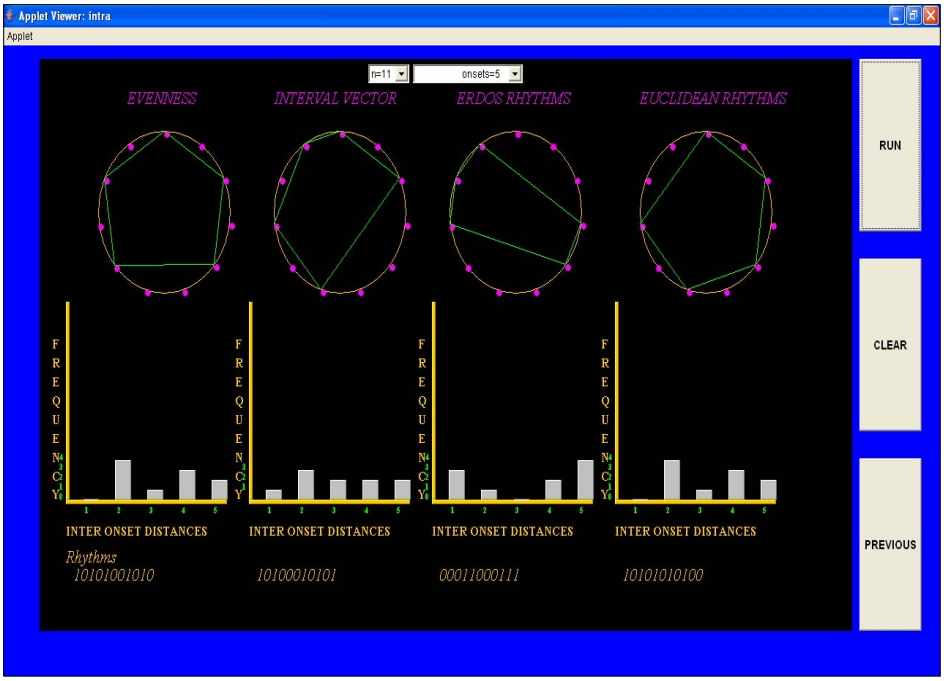
c. Erdos rhythms



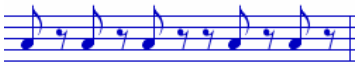
d. Euclidean rhythms



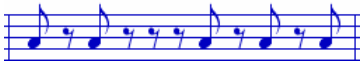
5. n =11, k=5



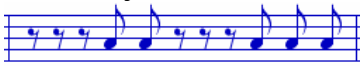
a. Evenness



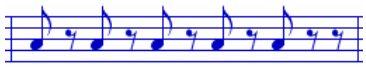
b. Interval vector



c. Erdos rhythms



d. Euclidean rhythms



4.1Sample Rhythms

Some sample rhythms with varying number of units (n) and onsets (k) are generated using Noteworthy Composer tool. They are posted on http://www.cs.uno.edu/~adlai/rhythms.

Chapter 5

CONCLUSION AND FUTURE WORK

Our attempt here was to develop a tool that generates rhythms using some of the methods and problems posed by Godfried Toussaint. The tool may be useful for a musical composer who can take some of the rhythms we have generated, analyze and insert them in music. In studying various rhythms from the past, evenness and the interval vector were two properties that were considered to analyze rhythms**.** Then, various methods were used to generate new rhythms that contain these properties to one degree or another.

Evenness is a property where onsets are distributed uniformly in a given time domain. The first procedure simply placed the onsets to form a regular *k*-gon inscribed in a circle where *k* is the number of onsets. This was achieved by placing one onset coincident with one lattice point and then moving the remaining onset points to their nearest lattice points in an evenly distributed way. The second procedure used the Bjorklund sequence generation algorithm which has the same structure as the well-known Euclidean algorithm. This procedure is based on computing the greatest common divisor which actually distributes the onsets uniformly.

A method known as Regression Evenness which measures the un-evenness in a rhythm was also implemented. The Regression Evenness Metric (REM) for a given rhythm measures how far the rhythm onsets are displaced from a perfectly even rhythm (with the same time frame and number of onsets) and is obtained by calculating the sum of all the onsets' deviations. The rhythm which yields a minimum REM value is called a maximally-even rhythm.

The interval vector is the spectrum of frequencies in a rhythm and is best represented by a histogram. The first procedure based on the interval vector efficiently generates rhythms that contain geometric constraints on their shapes in a given priority. The second procedure tried to achieve the Erdos-deep property by placing k onsets in a given time domain n such that, for each $i = 1, 2,..., k-1$, there is a non-zero distance that occurs exactly between i pairs from the set of k points.

Rhythms were generated with various combinations of values of n and k. These were then rendered in normal musical notation using a notation tool called NoteWorthy. The same software tool was then used to generate MIDI versions of the generated rhythms for purposes of actually listening to the generated rhythms. We even inserted some rhythms into existing musical works to listen and assess the aesthetic value of the generated rhythms. Eventually, of course, the listener will have to decide on whether a generated rhythm is "good" or bad. After all, beauty is in the eye of the beholder. Or in our case, in the ear of the listener.

As for future work, we can improve our system by developing an interface between our product and a sound producing tool such as NoteWorthy. The binary output of our system should be readily convertible into a MIDI format that can then become an input to any other MIDI-supported musical instrument. We can also envision our work to be incorporated into systems that are required to produce rhythmic patterns, such as keyboards and sequencers. This will, of course, require knowledge of algorithms, some of which might be proprietary, in order to achieve this.

REFERENCES

1) Godfried Toussaint, "The geometry of musical rhythm," *Proceedings of the Japan Conference on Discrete and Computational Geometry,* J. Akiyama et al. (Eds.), LNCS 3742, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 198-212.

2) Erik Demaine, Francisco Gomez-Martin, Henk Meijer, David Rappaport, Perouz Taslakian, Godfried Toussaint, Terry Winograd and David Wood, "The distance geometry of deep rhythms and scales," *Proceedings of the 17h Canadian Conference on Computational Geometry*, University of Windsor, Windsor, Ontario, Canada, August 10-12, 2005, pp. 160-163.

3)Godfried T. Toussaint, "The *Euclidean* algorithm generates traditional musical rhythms," *Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science*, Banff, Alberta, Canada, July 31 to August 3, 2005, pp. 47-56.

4) Godfried T. Toussaint, "Computational geometric aspects of musical rhythm," *Abstracts of the 14th Annual Fall Workshop on Computational Geometry*, Massachussetts Institute of Technology, November 19-20, 2004, pp. 47-48.

5) Godfried T. Toussaint, "A comparison of rhythmic similarity measures," *Proceedings of ISMIR 2004: 5th International Conference on Music Information Retrieval,* Universitat Pompeu Fabra, Barcelona, Spain, October 10-14, 2004, pp. 242-245.

6) Perouz Taslakian and Godfried T. Toussaint, "Geometric properties of musical rhythm," *Proceedings of the 16th Fall Workshop on Computational and Combinatorial Geometry*, Smith College, Northampton, Massachussetts, November 10-11, 2006.

7) Joao M. Martins, Marcelo Gimenes, Jonatas Manzolli, Adolfo Maia Jr. “Similarity Measures for Rhythmic Sequences”.

8) Miguel Diaz-Banez, Giovanna Farigu, Francisco Gomez, David Rappaport, and Godfried T. Toussaint, "El compas flamenco: A phylogenetic analysis," *Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science*, Southwestern College, Winfield, Kansas, July 30 to August 1, 2004, pp. 61-70.

9) Godfried T. Toussaint, "Algorithmic, geometric, and combinatorial problems in computational music theory," Extended version of paper that appeared in:
Proceedings of X Encuentros de Geometria Computacional, University of Sevilla, Sevilla, Spain June 16-17, 2003, pp. 101-107.

10) Godfried T. Toussaint, "A mathematical analysis of African, Brazilian, and Cuban *clave* rhythms," *Proceedings of BRIDGES: Mathematical Connections in Art, Music and Science,* Townson University, Towson, MD, July 27-29, 2002, pp. 157-168.

11) Greg Aloupis, Thomas Fevens, Stefan Langerman, Tomomi Matsui, Antonio Mesa, Yurai Nunez, David Rappaport and Godfried Toussaint, "Computing a geometric measure of the similarity between two melodies," *Proceedings of the 15th Canadian Conference on Computational Geometry*," Dalhousie University, Halifax, Nova Scotia, Canada, August 11-13, 2003, pp. 81-84.

12) Godfried Toussaint, "Mathematical features for recognizing preference in Sub-Saharan African traditional rhythm timelines," *Proceedings of the 3rd International Conference on Advances in Pattern Recognition*, University of Bath, Bath, United Kingdom, August 22-25, 2005, pp. 18-27.

13) Godfried T. Toussaint, "Classification and phylogenetic analysis of African ternary rhythm timelines," Extended version of paper that appeared in: *Proceedings of BRIDGES: Mathematical Connections in Art, Music, and Science*, University of Granada, Granada, Spain July 23-27, 2003, pp. 25-36.

14) Godfried T. Toussaint, "Algorithmic, geometric, and combinatorial problems in computational music theory," Extended version of paper that appeared in: *Proceedings of X Encuentros de Geometria Computacional*, University of Sevilla, Sevilla, Spain June 16-17, 2003, pp. 101-107.

15) D.H. Huson, "Splitstree: A Program for Analyzing and Visualizing Evolutionary Data," *Bioinformatics,* vol. 14, no. 10, pp. 68-73, 1998.

16) D. Bryant and V. Moulton, "NeighborNet: An Agglomerative Method for the Construction of Planar Phylogenetic Networks," *Algorithms in Bioinformatics, WABI 2002,* R. Guigó and D. Gusfield, eds., pp. 375-391, 2002.

17) B. Holland and V. Moulton, "Consensus Networks: A Method for Visualizing Incompatibilities in Collections of Trees," *Proc. Workshop Algorithms in Bioinformatics,* pp. 165-176, 2003.

18) http://www.noteworthysoftware.com/

19)Dannenberg, ``Listening to `Naima': An Automated Structural Analysis of Music from Recorded Audio," In *Proceedings of the 2002 International Computer Music Conference.* San Francisco: International Computer Music Association., (2002), pp. 28-34.

20) Dannenberg and Hu, ``Discovering Musical Structure in Audio Recordings," in *Music and Artificial Intelligence: Second International Conference*, C. Anagnostopoulo, M. Ferrand, A. Smail, eds., Lecture notes in computer science; Vol 2445: Lecture notes in artificial intelligence, Berlin: Springer Verlag, (2002), pp. 43-57.

APPENDICES

Appendix A: The Evenness Method

- In a rhythm, if the onsets on the circular lattice form a "regular polygon", then such a rhythm is an optimal rhythm.

- In this method the onsets (say k) are placed in the nearest positions to the actual positions (in case of a regular k-gon) such that it coincides with the instants of time into which the circular lattice is divided into.

- This gives a polygon which is an approximation of the optimal regular polygon.

- The rhythm generated is very close to the optimal rhythm.

```
public void calcFlow ()
{
//take the first input from the user and draw the circle
alg = true;
float y;             //geodesic distance (arc length) between
consecutive onsets when the onsets form a regular polygon
float z;             //specifies the position of the onset on
the circular lattice in case of a regular polygon

//temporary variables
int i;
int j;
int a;
float temp;

//initialisations
z=0;
for(i=1;i<nm;i++)
circle[i]=0;
circle[0]=1;          //starting with a high note-1 at time 0
y=nm/km;              //arc length between consecutive onsets in
a regular polygon - (k-gon)

//code for placing the onsets
//the onsets are placed in the nearest positions to the optimal
on the circular lattice.
while(z<nm)
{
z=z + y;         //actual position of onset
```

```
a=(int)z;
temp=z-a;
if(temp>0.5)    //Since the beats are recorded after every unit
interval of time the optimal position is rounded
a=((int)z)+1;   // off to its nearest position such that it
coincides with the instants of time when

//the beats are recorded.
//This gives a polygon which is an approximation of the regular
polygon.
circle[a] = 1;//an onset is placed at that approximated position
on the circular lattice.
}

//code to print the beats(1-high beat ,0-low beat) at different
instants of time
System.out.print("Rhythm generated - ");
for(i=0;i<nm;i++)
{
 System.out.print(circle[i]);
}

//Code for printing the frequencies of distances
cout<<endl<<endl;
cout<<"Frequency of occurrences of
distances(1......n/2)"<<endl<<endl;
for(i=0;i<1000;i++)
freqarray[i]=0;
int dist1;
int dist2;

//code to find all the inter-onset distances
for(i=0;i<nm;i++)
{
if(circle[i]==1)
{
for(j=i+1;j<nm;j++)
{
if(circle[j]==1)
                                        {
//there are 2 distances possible between 2 onsets on the
circular lattice
dist1=j-i;
dist2=nm-dist1;
```

```
//The shortest distance between 2 onsets is considered as the
distance between the onsets and its
// frequency of occurence is incremented.
if(dist1<dist2)
freqarray[dist1]++;
else
freqarray[dist2]++;
}
}
}
}
```

Appendix B:Euclidean Method

▪This method uses the euclid algorithm(it is an algorithm to find the greatest common divisor) and is also known as the bjorklund algorithm.

▪Find the difference between the two numbers and replace the largest number with the difference

▪Repeat this process till the difference is 1 or 0.The final number is the greatest common divisor.

```
public void calcFlow () {

alg = true;
int ones;               //number of onsets
int zeros;              //number of zeros
//the repeating part(sequence of beats 1-high beat, 0-low beat
of the rhythm.
intrepeat[]=newint[1000];
int n;                 //number of time intervals
int r;                 //the number of beats
//The number of times a part of the repeating part of the rhythm
occurs
int rem;
//lastzero specifies wether there is a single zero in the final
remainder column
char lastzero='n';

//temporary variables
int j;
int m1;
int m2;
int temp;
int c3;
int c5;
int tempvar;
int l1,l2;
int ty,r1;
for(int  cnt=0;cnt<1000;cnt++)
{
repeat[cnt]=0;
}
n=nm;
ones=km;
zeros=n-ones;
if(zeros>=ones)
{
m1=zeros;
```

```
m2=ones;
}
else
{
m1=ones;
m2=zeros;
}

//initially r represents the number of zeros that can be added
to the repeating rhythm
r=zeros/ones;

//rem is the number of remainder columns
rem=zeros%ones;

//the first note in the repeating rhythm is 1
repeat[0]=1;
if(rem!=0)
{
ty=r+1;
}
else
{
ty=r;
}
for(j=1;j<=ty;j++)
{

//implementing euclidean algorithm. The difference m1-m2 gives
the number of remainder columns
m1=m1-m2;
repeat[j]=0;
}
m1=m1+m2;
if(m1>=m2)
temp=m1-m2;
else
temp=m2-m1;
if(rem==1)
{
lastzero='y';
j--;
r=rem-1;
temp=1;}
if(rem!=1)
{
if(zeros>=ones)
```

```
{
temp=m2-m1;
m2=temp;
}
else
{
temp=m1-m2;
m1=temp;
}
}
if(rem==0)
temp=0;
j--;
while(temp!=1&&temp!=0){
if(temp/rem!=0)
{
for(int cnt2=0;cnt2<=r;cnt2++)
{
j++;
l1=repeat[cnt2];
repeat[j]=l1;
}
if(m1<=m2)
{
temp=m2-m1;
m2=temp;
}
else
{
temp=m1-m2;
m1=temp;
}

//if the number of remainder columns is 1 or 0 then break
if(temp==1||temp==0)                                        break;}
else
{
r1=j;
for(int cnt2=0;cnt2<=r;cnt2++)
{
j++;
l2=repeat[cnt2];
repeat[j]=l2;
}
rem=temp;
r=r1;
if(m1<=m2)
```

```
{
temp=m2-m1;
m2=temp;
}
else
{
temp=m1-m2;
m1=temp;
}
if(temp==1||temp==0)
break;
}
}

//Code to assign the rhythm to the circlefinal array
//initialisation
for(c3=0;c3<1000;c3++)
{
freqarray[c3]=0;
circlefinal[c3]=0;
}
int cnt5;
int cnt6;
c3=0;

//if the number of remainder columns is 0 then the final rhythm
is the
//repeating part of the rhythm occuring a certain number of
times.
if(temp==0)
{
for(int cnt3=0;cnt3<m1;cnt3++)
{
for(int cnt4=0;cnt4<=j;cnt4++)
{
circlefinal[c3]=repeat[cnt4];
c3++;
}
}}

//if the number of remainder columns is 1 then the finaal rhythm
is the
//repeating part of the rhythm occuring a certain number of
times
//followed by the sequence of beats in the remainder columns.
else if(temp==1)
{
```

```
if(m1>m2)
cnt5=m1;
else
cnt5=m2;
for(int cnt3=0;cnt3<cnt5;cnt3++)
{
for(int cnt4=0;cnt4<=j;cnt4++)
{
circlefinal[c3]=repeat[cnt4];
c3++;
}
}

//code to add the sequence of beats in the remainder column.
cnt6=r;
for(int cnt7=0;cnt7<=cnt6;cnt7++)
{
if(lastzero=='n')
{
circlefinal[c3]=repeat[cnt7];
c3++;
}
elseif(lastzero=='y')
//lastzero=y if the remainder column has a single zero
{
circlefinal[c3]=0;
c3++;
}
}}
c5=0;
for(c5=0;c5<n;c5++)
{
tempvar=circlefinal[c5];
}

//code for printing the frequencies of distances
int dist1;
int dist2;
for(c5=0;c5<n;c5++)
{
if(circlefinal[c5]==1)
{
for(int c6=c5+1;c6<n;c6++)
{
if(circlefinal[c6]==1)
{
```

```
//there are 2 distances possible between 2 onsets on the
circular lattice
dist1=c6-c5;
dist2=n-dist1;

//The shortest distance between 2 onsets is considered as the
distance between the onsets and
//its frequency of occurence is incremented.
if(dist1<dist2)
freqarray[dist1]++;
else
freqarray[dist2]++;
}
}
}
}
}
```

Appendix C: Comparison Measure

```
#include <iostream>
using namespace std;
int main()
{
//array storing the values (1-high note or 0-low note) at
different instants of time
int circle[1000];
//number of time intervals into which the circle is divided
float n;
//number of onsets
float ones;
//geodesic distance (arc length) between consecutive onsets when
the onsets form a regular polygon
float dist;
//deviation of the onset fro its optimal position on the cirular
lattice
float deviation;
//sum of the deviations of all the onsets
float sumofdev=0;
//temporary variables
float k;
float temp;
cout<<endl<<"circular lattice number=";
cin>>n;
cout<<endl<<"the number of onsets=";
cin>>ones;
dist=n/ones;
cout<<endl<<"enter the rhythm"<<endl;
for(int i=0;i<n;i++)
cin>>circle[i];

//Code to find the net deviation of the rhythm from the optimal.
//This net deviation is called the regression evenness of the
rhythm.
k=0;

//j is the position of the onset in the given rhythm
for(int j=0;j<n;j++)                    {
if(circle[j]==1){
//Corresponding optimal position of the onset on the circular
lattice in case of a regular polygon
temp=k*dist;if(temp>j)
//deviation of the position of the onset in the given rhythm
from its optimal position
deviation=temp-j;
```

```
else
deviation=j-temp;
sumofdev=sumofdev+deviation;
k++;
}
}
cout<<endl<<"total deviation="<<sumofdev;
return 0;
}
```

Appendix D: Interval vector method

```
public void calcFlow () {

int i,j;
int startdist;
int gap;
int alrprones;
int circle[]=new int[1000];
int array[]=new int[1000];
int maxcover;
int maxfreq=0;
int zerotemp;
int k1,k2;
int value;
char gapres='n';
int count=0;
int tmparr[]=new int[1000];
int filled='n';
int d1,d2;
int mindist;
int c1;
int c2=0;
int c3;
int c4=0;
int val=0;
char allgaps='y';
int numgaps=0;
int numconv;
int arraytemp[]=new int[1000];
int conversions;
int sum;
int flatsum;
int gaponcircle;
int circlegap=0;
//int beforeval;
float midpoint1;
//float midpoint2;
float midpnt=0;
float diff1;
float diff2;   //copy every constant corr to that onset
int maxcovdist=0;
int finalcount;
gapfinal=1000;
zerofreq=1000;
maxfinal=1000;
difffinal=1000;
```

```
maxgapfinal=1000;
for(finalcount=1;
finalcount<=nm/2;finalcount++)
{
again='n';

//initialisation
alrprones=1;
startdist=1;
k1=1;
for(i=0;i<1000;i++)
{
array[i]=0;
circle[i]=0;
}
circle[0]=1;
circle[nm-finalcount]=1;
array[finalcount]++;
for(i=1;i<=km-2;i++)
{
allgaps='y';
conversions=0;
maxfreq=1000;
flatsum=10000;
gaponcircle=1000;
numgaps=1000;
conversions=0;
for(j=0;j<nm;j++)
{
if(circle[j]==0)
{
value=j;
filled='n';
count=0;
gap=0;
gapres='n';
sum=0;
numconv=0;
for(k2=0;k2<1000;k2++)
arraytemp[k2]=array[k2];
for(k2=0;k2<nm;k2++)
{
if(circle[k2]==1)
{
if(value>k2)
d1=value-k2;
else
```

```
d1=k2-value;
d2=nm-d1;
if(d1<d2)
mindist=d1;
else
mindist=d2;
count++;
tmparr[count]=mindist;
if(arraytemp[mindist]==0)
numconv++;
arraytemp[mindist]++;
sum=sum+arraytemp[mindist];
}
}
maxcover=0;
for(c1=1;c1<=count;c1++)
{if(tmparr[c1]>maxcover)
maxcover=tmparr[c1];
}
gapres='n';
for(c1=1;c1<=(nm/2);c1++)
{if(arraytemp[c1]==0 && gapres=='n')
{
gap++;
gapres='y';
}
else if(arraytemp[c1]==1)
{
gapres='n';
}
}
if(arraytemp[nm/2]==0)
gap--;
int maxm=0;
for(c2=1;c2<=maxcover;c2++){
if(arraytemp[c2]>maxm)
maxm=arraytemp[c2];
}
for(c2=0;c2<=nm;c2++)
{
if(circle[c2]==1||c2==nm)
{if(c2>value)
{
circlegap=c2-c4;
break;
}c4=c2;
}
```

```
}
if(gap<numgaps)
{
val=value;
numgaps=gap;
conversions=numconv;
maxfreq=maxm;
flatsum=sum;
gaponcircle=circlegap;
midpnt=midpoint1;
maxcovdist=maxcover;
}
else if(gap==numgaps)
{if(numconv>conversions)
{
val=value;
conversions=numconv;
maxfreq=maxm;
flatsum=sum;
gaponcircle=circlegap;
midpnt=midpoint1;
maxcovdist=maxcover;
}
else if(numconv==conversions)
{

//third concept
if(maxm<maxfreq)
{
val=value;
maxfreq=maxm;
flatsum=sum;
gaponcircle=circlegap;
midpnt=midpoint1;
maxcovdist=maxcover;
}
else if(maxm==maxfreq)
{if(sum<flatsum)
{
val=value;
flatsum=sum;
gaponcircle=circlegap;
midpnt=midpoint1;
maxcovdist=maxcover;
}
else if(sum==flatsum)
```

```
{
if(circlegap>gaponcircle)
{
val=value;
gaponcircle=circlegap;
midpnt=midpoint1;
maxcovdist=maxcover;
}else if(circlegap==gaponcircle)
{
if(midpoint1>value)
diff1=midpoint1-value;
else diff1=value-midpoint1;
if(midpnt>val)
diff2=midpnt-val;
else diff2=val-midpnt;
if(diff1<diff2)
{
val=value;
midpnt=midpoint1
maxcovdist=maxcover;
}else if(diff1==diff2)
{//do nothing
}}}}}
count=0; }}
for(c2=0;c2<nm;c2++)
{
if(circle[c2]==1)
{
if(val>c2)
d1=val-c2;
else
d1=c2-val;
d2=nm-d1;
if(d1<d2)
mindist=d1;
else
mindist=d2;

//Compromise if all poss dist are covered then check for all
uncovered points
array[mindist]++;
}
}
circle[val]=1;
startdist=maxcovdist+1;
```

```
//if no: of gaps is same then more no: of zeros convert to ones
}//for each onset placed
zerotemp=0;
for(c3=1;c3<=(nm/2);c3++)
{
if(array[c3]==0)
zerotemp++;
}
minfreq=1000;
for(c3=1;c3<=nm/2;c3++)
{
if(array[c3]<minfreq && array[c3]!=0)
minfreq=array[c3];
}
freqdiff=maxfreq-minfreq;
maxgap=0;
maxgaptemp=0;
gapresf='n';
for(c3=0;c3<=nm;c3++)
{
if(circle[c3]==1||c3==nm)
{
if(maxgaptemp>=maxgap)
maxgap=maxgaptemp;
maxgaptemp=0;
}
else if(circle[c3]==0)
maxgaptemp++;
}
//conditions
if(numgaps<gapfinal)
{
gapfinal=numgaps;
again='y';
}

//initialisation
else if(numgaps==gapfinal)
{
if(zerotemp<zerofreq)
{
again='y';
zerofreq=zerotemp;
}
else if(zerofreq==zerotemp)
{
if(maxfreq<maxfinal)
```

```
{
again='y';
maxfinal=maxfreq;
}
else if(maxfreq==maxfinal)
{
if(freqdiff<difffinal)
{
again='y';
difffinal=freqdiff;
}
else if(freqdiff==difffinal)
{
if(maxgap<maxgapfinal) //find maxgap
{
again='y';
maxgapfinal=maxgap;}
else if(maxgap==maxgapfinal)
{//do nothing
}}}}}
if(again=='y')
{
for(c3=0;c3<nm;c3++)
{
if(circle[c3]==0)
circlefinal[c3]=0;
else
circlefinal[c3]=1;
arrayfinal[c3]=0;
}
for(c3=1;c3<=(nm/2);c3++)
arrayfinal[c3]=array[c3];

//other properties
gapfinal=numgaps;
zerofreq=zerotemp;
maxfinal=maxfreq;
difffinal=freqdiff;
maxgapfinal=maxgap;
}
}
alg = true;
System.out.print("rhyyythm   ");
for(int i1=0;i1<nm;i1++)
System.out.print(""+circlefinal[i1]);
}
```

Appendix E: Erdos-deep method

```
public void calcFlow () {
int circle[]=new int[1000];
int m=1;
int i,j,k1;
int temp;
char nextpart='n';
char gotit='n';
int primeval=(3*nm)/4;
int finalnumofsem=10000;
while(gotit=='n')
{
for(i=0;i<nm;i++)
{
circle[i]=0;
arrayfinal[i]=0;
}
if(nextpart=='n')
{
for(i=primeval;i>=2;i--)
{
if(gcd(i,nm)==1)
{
m=i;
break;
}
}
}
if(i<2)
{
primeval=((3*nm)/4)+1;
nextpart='y';
}
if(nextpart=='y')
{
for(i=primeval;i<nm;i++)
{
if(gcd(i,nm)==1)
{
m=i;
break;}}}
if(i==nm)
{
```

```
//all the relative prime numbers are checked
gotit='y';
break;
}
else
{
if(nextpart=='n')
primeval=m-1;
else
primeval=m+1;
for(i=0;i<km;i++)
{
temp=(m*i)%nm;
circle[temp]=1;
}

//checking for semicircles
int numofsemi=0;
int prev=0;
int current;
int curr;
for(curr=1;curr<=nm;curr++)
{
if(curr==nm)
current=0;
else
current=curr;
if(circle[current]==1)
{
if(prev<current)
temp=current-prev;
else
temp=nm-(prev-current);
if(temp>(nm/2)+1)
{
numofsemi=numofsemi+(temp-(nm/2)-1);
     }
prev=current;
}
}
if(numofsemi<finalnumofsem){
finalnumofsem=numofsemi;
for(i=0;i<nm;i++)
{
if (circle[i]==1)
circlefinal[i]=1;
else
```

```
circlefinal[i]=0;
}
}
if(finalnumofsem==0)
{
gotit='y';
break;
}
}
}
for(j=0;j<nm;j++)
{
if(circlefinal[j]==1)
{
for(k1=j+1;k1<nm;k1++)
{
if(circlefinal[k1]==1)
{
if((nm-(k1-j))>(k1-j))
temp=k1-j;
else
temp=nm-(k1-j);
arrayfinal[temp]++;
}}}}
alg = true;
}
int gcd(int a,int b)
{
if(a==b)
{
return a;
}
else
{
if(a>b)
return gcd(a-b,b);
     else
return gcd(a,b-a);
}}
```

Vita

Shashidhar Sorakayala was born in the city of Hyderabad in India, on May 27th, 1982. He got his Bachelors of Technology in Computer Science from Jawaharlal Nehru Technological University in May, 2003. He joined the Masters of Sciences in Computer Science at the University of New Orleans in 2004. During this time, he worked as a student worker in Chemistry Department.