

GENETIC ALGORITHMS AS A MODEL OF MUSICAL CREATIVITY – ON GENERATING OF A HUMAN-LIKE RHYTHMIC ACCOMPANIMENT

Martin DOSTÁL

*Department of Computer Science
Faculty of Natural Sciences
Palacký University
Tomkova 40
779 00 Olomouc, Czech Republic
e-mail: dostal@inf.upol.cz*

Manuscript received 10 January 2005
Communicated by Vladimír Kvasnička

Abstract. This article introduces a genetic algorithm based system intended for automated generating of a realistic rhythmic (drum set) accompaniment. Present systems do not insist on the natural music criteria and realistic (human-like) result. They generate a rhythmic accompaniment regardless to the other instruments used. The fitness operators are mostly based on manual evaluation by user. The system described in this paper uses automatic fitness evaluator and prefers some of the natural music criteria. Accompaniment is generated with regard to a *harmonic-accompaniment instrument* (HAI).

Keywords: Genetic algorithms, evolutionary music systems, rhythm, digital arts

1 INTRODUCTION

Modern artificial intelligence techniques are (among other things) used in many human domains like aesthetics and art. There are some AI-based systems for generating of visual art (e.g. AARON) or music (GenJam, CONGA). The genetic algorithm (GA) based artificial music systems are mostly applied to generate a harmonic accompaniment. These systems “hear” the melody and they should produce an “interesting” and criteria satisfying harmony. However, this task is quite far

from the generating of rhythms, because theory of music harmony is more subtle, so that there can be found enough explicit rules for this task. In the case of musical rhythm, there is *a fundamental rule* – the so called “skeleton of rhythm” – *characteristic rhythm figure*. This concept will be the starting point for our investigations. Every concrete accompaniment (of some rhythm) contains characteristic rhythmic figure and its adaptations or extensions. However, we can’t explicitly describe these adaptations (extensions) by explicit and sufficiently distinct rules. In a musical reality, the form and the realisation of rhythmic accompaniment was much more influenced by “usage” (evolution of musical styles, instrumentation tradition) than some distinct or explicit rules. We assume that other than rule-based approach is more appropriate to apply to generating a rhythmic accompaniment.

1.1 Genetic Algorithms and Rhythm

What relation is there between search algorithms and a rhythm composition? We can imagine the music composition process as a search problem. The search space contains all possible compositions (all combinations) and a composer (player) is a “searcher” who composes – searches the searching space for a composition which satisfies his/her criteria (idea, feeling, intention). It can be said that talented and musically more erudite composer has a “higher chance” (probability) to achieve better composition than a musically less erudite composer using the same search effort.

1.2 Genetic Algorithm as a Model of Rhythm Creativity

The composer takes an advantages of his/her invention – the ability to produce an original idea, and creativity – the ability to transform, adapt and apply various knowledge, ideas and thoughts. There is no distinct proof that we can effectively simulate a human-like musical invention, but we are able to “reprint” the result of a human invention into the machine. It can be done for instance by using the population of GA. Note that machine is able to learn a musical knowledge as well as people do. Crossover and mutation serves here for the transformation and adaptation of musical inventions (elements of population) – *GA could be understood as a model for (artificial) rhythm creativity*.

1.3 Previous Work on GAs and Rhythms

The first known system which outlined the usage of GA for the generating of rhythms was the Damon Horowitz’s system [5] which worked with a randomly generated initial population of 10 chromosomes. The result of GA was a chromosome – rhythmic phrase – one beat long rhythm pattern, which represents (only) one percussion instrument so that the resultant pattern is not polyphonic. Both crossover and mutation operators worked randomly without any musical knowledge. The fitness operator is based on manual evaluation by the user.

Second interesting approach can be found in [10] where the system called *CONGA* is described. *CONGA* produces a rhythmic phrase from 4 to 16 beats long. The population is constructed using two-dimensional chromosomes. Columns determine the volume (strictly speaking the velocity) of a drum instrument, and lines determine an individual drum instruments staff. The initial population contains sample rhythmic phrases which are one beat or half-beat long. The *GONGA*'s fitness operator is based on manual evaluation by the user and it is extended with a three-layer neural network which learns responses from the user and works as an evaluation assistant.

Both described systems generate rhythmic phrases as stand-alone boxes (regardless to the other instruments). The *GeneticDrummer* system described here generates a matching rhythmic accompaniment (optionally with a break) to an arbitrarily long harmonic accompaniment (like bass or rhythm piano) phrase. For instrumentation a conventional drum set is used, composed of *snare drum*, *bass drum*, four *toms*, *hi-hat*, two *crash cymbals*, *splash* and *ride cymbal*. The generating of the accompaniment is based (by design) on *local restriction* – GA can “hear” only current beat of the harmonic accompaniment to the rhythm phrase which is generated. However, there is a global (indirect) relation ensured by the harmonic accompaniment phrase itself.

2 DESIGN OF THE GENETIC ALGORITHM

Now we will describe the design of the used GA. It partially stems out from the well known basic genetic algorithm called *Simple Genetic Algorithm* (SGA). The main differences and innovations can be found in the construction of the population and in the design of the fitness and mutation operators.

2.1 Population

The population carries a musical knowledge about the concrete rhythm. The elements of the population (the so-called *multichromosomes*, which are described below) are rhythmic figures (grooves) – concrete examples of concrete rhythm. In other words, we can imagine the population as a set of drum practice phrases that the player is able to play, so that *learning of GeneticDrummer (genetic algorithm) is similar to the human drummer*. We can “teach” *GeneticDrummer* by “rewriting” practices from the drum playing schoolbooks into the population. The rhythm figures are always one beat long.

The population of GA reflects *the human player's musical knowledge* (the characteristic grooves). The used GA contains two types of population – *the accompaniment (groove) population* for generating an accompaniment and *the break population* for generating rhythm breaks (usually between grooves) or solos. The initial populations should be musically consistent and should represent right grooves and breaks. The populations can hold arbitrary even number of multichromosomes. The population construction is defined by the following rules:

1. A basic item is *the rhythm element*
2. Rhythm elements construct *chromosomes*
3. Chromosomes construct *multichromosomes*
4. Population is a set of a multichromosomes.

2.1.1 Rhythm Elements

The basic musical symbol is generally a note¹. We assume that note is *not optimal* as basic element for realistic, flexible and natural representation of rhythm instrumentation. Natural drum playing utilises (naturally except single stroke) the so called *multiple drum strokes* (see Figure 1), when a player is able to produce a couple of notes by playing one stroke only. We use these single or multiple strokes as the basic elements and we call them the *rhythm elements*.

A similar approach to our rhythm elements can be found in the common drum notation – the multiple strokes are written as one note with a special symbol. The usage of the rhythm elements instead of notes allows for more expressive result – an element can be simply replaced by another element (for instance using a mutation operator). The rhythm elements are more handy and useful for the representation of natural drum playing nuances (multiple strokes) than single notes. The composition of multiple strokes using single notes (by some rules for example) without musical flaws would be disproportionately difficult.

The rhythm elements are individually coded for each instrument (drum) to allow to represent the natural technical possibilities and the specific features of each represented drum (or cymbal) in a drum set. For instance: we can play a drumhead stroke or a side-stick stroke on snare drum but only a drumhead stroke on bass drum; or we can stroke hi-hat opened, closed or foot splashed, but not the other drums or cymbals.

2.1.2 Chromosome

Every chromosome represents one instrument (one staff-line on the staff). The length of the chromosome and the specified chromosome resolution determines the metre. Every chromosome element (gene) represents a note (stroke) or a rest with the duration corresponding to the given resolution (for example: for sixteenth-note resolution one gene (element) represents one sixteenth note). Interpretation of the genes is described in the so-called *implementation tables*, see Tables 2–4 for the example of the implementation tables for the bass drum, the snare drum and the hi-hat, respectively (implementation tables for other drums and cymbals are not given here due to lack of space). The first digit of the value from the implementation table determines the *stroke type* and the second digit determines the *stroke velocity*. The special code 00 means a current-resolution-long rest and the special definition (**empty**) means empty chromosome (the chromosome composed of the rests).

¹ It is also the basic element for the both above-mentioned systems.

Note symbol	Meaning	Element name
		single stroke
		double stroke
		triple stroke
	 <i>mp</i>	flam
	 <i>o < mp</i>	drag
	 <i>o < mp</i>	ruff

Table 1. Rhythm elements

Playing technique	<i>p</i>	<i>mp</i>	<i>mf</i>	<i>f</i>	<i>ff</i>	<i>fff</i>
Single stroke	11	12	13	14	15	16

Table 2. Implementation table – bass drum

2.1.3 Multichromosome

Multichromosome is a list composed of the chromosomes. The structure is as follows:

- ((chromosome representing splash cymbal)
- (chromosome representing crash cymbal 2)
- (chromosome representing crash cymbal 1)
- (chromosome representing ride cymbal)

Playing technique	<i>p</i>	<i>mp</i>	<i>mf</i>	<i>f</i>	<i>ff</i>	<i>fff</i>
single stroke	11	12	13	14	15	16
double stroke	21	22	23	24	25	26
triple stroke	31	32	33	34	35	36
flam	41	42	43	44	45	46
drag	51	52	53	54	55	56
ruff	61	62	63	64	65	66
side-stick	71	72	73	74	75	76

Table 3. Implementation table – snare drum

Playing technique	<i>p</i>	<i>mp</i>	<i>mf</i>	<i>f</i>	<i>ff</i>	<i>fff</i>
hi-hat closed, single stroke	11	12	13	14	15	16
hi-hat closed, double stroke	21	22	23	24	25	26
hi-hat closed, triple stroke	31	32	33	34	35	36
hi-hat closed, flam	41	42	43	44	45	46
hi-hat closed, drag	51	52	53	54	55	56
hi-hat closed, ruff	61	62	63	64	65	66
hi-hat open, single stroke	71	72	73	74	75	76
hi-hat foot	81	82	83	84 </td <td>85</td> <td>86</td>	85	86

Table 4. Implementation table – hi-hat

(chromosome representing hi-hat)
(chromosome representing snare drum)
(chromosome representing bass drum)
(chromosome representing small tom-tom)
(chromosome representing medium tom-tom)
(chromosome representing large tom-tom)
(chromosome representing floor-tom))

Example of a multichromosome and the corresponding notation is shown in Figure 1:

((empty)
(empty)
(empty)
(empty)
'(14 00 14 00 14 00 14 00 44 00 74 00 24 00 15 00)
'(00 00 00 00 14 00 00 00 00 00 00 00 14 00 00 00)
'(14 00 00 00 00 00 14 00 14 00 00 00 00 00 00 00)
(empty)
(empty)
(empty)
(empty))

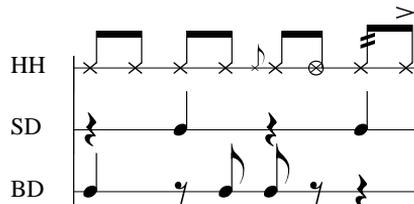


Fig. 1. Example of notation

2.2 Selection Operator

Selection of the multichromosomes is based on a random selection of the parental multichromosomes. All multichromosomes have the same selection probability.

There are two different selection strategies:

Arbitrary selection – pairs of multichromosomes are selected randomly, the selected multichromosomes are not deleted from the list of selected multichromosomes, so that one multichromosome can be selected more than once.

Strict selection – every selected multichromosome is deleted from the selection list; thus, every multichromosome will be selected exactly once.

2.3 Crossover Operator

The multichromosomes are crossed at one randomly selected position (every chromosome in the multichromosome at the same position). The crossover operator produces a pair of new multichromosomes (offsprings) in the same way as SGA do.

2.4 Mutation Operator

The mutation operator is composed of several functions called *mutation operators*. The mutation operators are applied to the chromosomes. They serve as an efficient way to fetch more indeterminism to the GA and to adapt multichromosomes to acquire higher fitness. The mutation operators are defined generally and they can be applied to arbitrary chromosomes. The configuration of the mutation operators is stored in the structure called *mutation script*. Every instance of the mutation operator has assigned a weight of the operator. The weight of the operator is defined stochastically using a random “lot” from interval $< 0, 1 >$, which determines the probability that the operator will be applied on the current rhythm element (for example: 1 means absolute certainty, 0.5 means 50% probability, 0 means no possibility).

2.4.1 Accentuate Operator

It imitates velocity accents (rhythm phrasing) of the accompaniment harmonic instrument (HAI). It is useful for the generating of breaks, solos or bass drum patterns.

Scheme:

```

DO for all chromosomes of a multichromosome:
  DO for all elements of a chromosome:
    IF current element is in the rhythm consonance with HAI AND
      the volume intensity of the current element is less than 4 AND
      a lot is positive THEN
      add 2 to the volume intensity of the rhythm element.
```

2.4.2 Syncopize Operator

It imitates a simple syncopation effect². Syncopation is useful in music genres like jazz or latino. For adequate results, it is recommended to set the operator weight to a relatively small value.

Scheme:

```
DO for all chromosomes of a multichromosome:
  DO for all elements of a chromosome:
    IF the current element stands on strong beat AND
      a lot is positive THEN
      locate syncopable positions for the current element
      AND randomly select a syncopable position
      IF volume intensity > 1 THEN
        decrement volume of the rhythm element
      DO syncopation (move element to the selected position)
```

2.4.3 Rhythm-Element-Change Operator

It changes the type of the rhythm element (stroke type).

Scheme:

```
DO for all chromosomes of a multichromosome:
  DO for all elements of a chromosome:
    IF a lot is positive THEN
      randomly change the type of element AND
      the volume intensity (velocity) of element left unchanged
```

2.4.4 Bass-Teamwork Operator

This operator ensures (adapts) rhythm consonance between a chromosome (drum) and HAI. It is especially used to generate drum breaks or bass drum patterns rhythmically consonant to HAI. Because the bass drum usually does not play³ on the strong beats, this operator skips rhythm elements standing on the strong beats.

Scheme:

```
DO for all chromosomes of a multichromosome:
  DO for all elements of a chromosome:
```

² File with population definition contains a list of the strong beats in population's rhythm which is a necessary parameter for this operator.

³ An instance of exception from this assertion may be some double bass drum patterns in rock music, where the bass drum may be played on the strong beats as well. In such cases it is recommended to set the weight of the operator to a negligible number.

```

IF there is on the current position in chromosome any rhythm
  element, but no HAI note THEN
  delete current rhythm element (replace it with code 00)
IF the current rhythm element stands on the strong beat
  in the current measure THEN delete it
IF there is a note on the current position of HAI,
  but no rhythm element THEN
  place at the current position rhythm element 14

```

Mutation script example:

```

'(list
  (list 'notes-bass 'mutation-bass-teamwork 0.12 music)
  (list 'notes-snare 'mutation-syncopize 0.102 music)
  (list 'notes-snare 'mutation-rhythm-element-change 0.029))

```

2.5 Fitness Operator

The fitness operator is based on a similar idea as the mutation operator. There are several fitness operator functions (fitness operators) which can be applied on arbitrary chromosome(s) in a multichromosome. Fitness operators evaluate quantitative parameters of generated rhythmic accompaniment. Qualitative parameters (like drive, or tastiness) can be objectively evaluated by a human only, but the design of evaluation of the quantitative parameters by the fitness operators establishes a fine assumption of meeting the quality criteria.

The global fitness setup is saved in the so-called fitness script. Some of the fitness operators use HAI for the evaluation and the others use user defined criteria – the user can specify the so-called *ideal effect*. The main objective of *ideal effect* is to formulate quantitative characteristics of the result desired by the user. Ideal effect is a number from interval $< 0, 1 >$, which describes the operator result desired by the user. 0 means minimum result and 1 maximum result. For instance: ideal effect set to 1 on Rhythm-quantity operator means that the best chromosome (with highest fitness) will be considered a chromosome with all positions set to a non-zero rhythm element. The resultant value is normalized by formula (1), so that normalized value is from interval $< 0, 1 >$.

$$q_o = 1 - \left| i_u - \frac{f}{f_{max}} \right| \quad (1)$$

$$q_o = \frac{f}{f_{max}} \quad (2)$$

where: f – acquired fitness value, f_{max} – maximum achievable fitness value, i_u – ideal effect.

2.5.1 Rhythm-Conformity Operator

This operator evaluates rhythm consonance between an HAI and a chromosome. The rhythm consonance characterises a situation when both current HAI note and the chromosome element stand on the current position. The value is normalized by formula (2), so that this operator returns the relation between acquired and achievable rhythm consonance.

2.5.2 Rhythm-Unconformity Operator

This operator evaluates rhythm dissonance between an HAI and a chromosome. The rhythm dissonance characterises a situation when the HAI note stands on the current position but the rhythm element does not. The value is normalised by formula (2), so that this operator returns relation between acquired and achievable rhythm dissonance.

2.5.3 Dynamic-Conformity Operator

This operator evaluates the relation of velocity intensity between an HAI and a chromosome. Every element of the chromosome is evaluated according to formula (3), where c_l describes a chromosome length (normalisation parameter), N_{dhn} is a set of an HAI notes, e_i is the i^{th} rhythm element, and dhn_i is a i^{th} HAI note.

$$d_c = \frac{1}{c_l} \sum_{i \in N_{dhn}} \left(1 - \frac{|e_i - dhn_i|}{dhn_i}\right) \quad (3)$$

2.5.4 Instrument-Quantity Operator

This operator evaluates a weighted count of rhythm elements in a chromosome. The weights of rhythm elements are specified in Table 5. The resultant value is normalised by formula (1).

Element type	Value
single stroke	1.1
double stroke	1.2
triple stroke	1.3
flam	1.2
drag	1.3
ruff	1.4

Table 5. Weights of the rhythm elements

2.5.5 Instrument-Quantity-on-Strong-Beats Operator

This operator is similar to the *Instrument-quantity* operator except that it evaluates only quantity on the strong beats of a multichromosome. The resultant value is normalised by formula (1).

Fitness operator parameters are specified in the fitness script: weight – arbitrary whole number (suggested range is $< -100, 100 >$) and a chromosome (instrument) on the operator to be applied, and (if supported by the operator) the ideal effect. An example of the fitness script can be seen on page 333. Note the ideal effect specified (the rightmost value in a given row) on instrument-quantity operator in this example. The following formula (4) computes total fitness value.

$$fitness = \sum_{k=1}^n w_k f_k \quad (4)$$

where: w_k specifies operator weight and f_k fitness of the k^{th} fitness operator. If the resultant fitness is less than 1, then it is set to be 1 (resultant fitness should not be zero or negative).

2.6 Reproduction Operator

Reproduction of multichromosomes is done by the well known *weighted roulette-wheel* operation. Each multichromosome belongs to the part of the roulette so that a multichromosome with higher fitness has higher probability to be reproduced into the new generation.

2.7 Final Selection

After GA passes the given count of generations then the best multichromosome is returned from the current population as the result.

3 FUNCTIONAL SCHEME OF GA

Functional scheme of the described system will be described in the following parts:

1. scheme for generating accompaniment of the *current* measure (beat)
2. scheme for generating whole accompaniment phrase.

3.1 Generating the Current Beat

One run of GA produces the matching drum set accompaniment (optionally ended by break) to the current (one) beat of HAI.

Scheme:

1. evaluate initial population
2. reproduce initial population
3. make selection
4. make crossover
5. mutate new population (population of offsprings)
6. evaluate new population
7. reproduce new population
8. if the algorithm passes given count of generations, then go to step 9, else generate next generation: repeat from step 1
9. select final multichromosome (winner) – produced accompaniment

3.2 Generating the Whole Phrase

1. read HAI phrase from the MIDI file
2. read fitness script for groove and break population
3. read mutation script for groove and break population
4. set current beat
5. load population for a groove
6. generate current beat (see the scheme below)
7. add generated beat to the other generated beats
8. while generating of the accompaniment not completed, then repeat from step 4
9. if the system is set to generate break then continue, else go to the end
10. set current beat to the last beat
11. load population for a break
12. read fitness script
13. read mutation script
14. generate a beat
15. according to the user preferences append the result to the correct position.

4 SIMULATING THE PSYCHOMOTORIC PERFORMANCE IMPERFECTION OF A HUMAN PLAYER

It is well known that a human player does not play with absolute accuracy. These imperfections are not perceived as a mistake, rather as a fine drive of human player instrumentation. We involved this into the system by implementating of the so called Humanizer module which adds imperfections to generated rhythmic accompaniment

(the time-shifting and velocity-shifting of rhythm elements). Time-shifting is implemented as a random untuning of the time position of notes. Maximal time untuning is determined by time-shift parameter (up to $\pm\frac{1}{32}$ note). Velocity-shifting untunes the velocity (volume) of notes. Maximal velocity untuning is determined by velocity-shift parameter (up to $\pm 25\%$). The sliders for customizing the Humanizer can be seen on the screenshot of user interface in Figure 2.

5 IMPLEMENTATION AND USAGE OF THE DESCRIBED SYSTEM

The described system (we call it GeneticDrummer) was implemented in LISP, including GUI. GeneticDrummer works in conjunction with professional hard disc recording (HDR) applications Cakewalk ProAudio or Cakewalk Sonar. The user selects an HAI phrase⁴ in a HDR application and starts the rhythmic accompaniment generating process from GeneticDrummer GUI, where the parameters of genetic algorithm can be set up (populations, number of generations), parameters of Humanizer and generated phrase: the user can select whether a generated phrase will start with a crash cymbal stroke⁵ and starting beat of last measure, where a break will be generated. The generated accompaniment is returned to HDR application as a new MIDI track. Such track can be further user-edited, as well as the other tracks. Screenshot of GUI can be seen in Figure 2.

6 EMPIRICAL MEASUREMENTS OF GA

Now some empirical measurements will be shown. Complex and rigorous measurement would be quite problematic, because results depend on many circumstances – used HAI, user-specified criteria, used mutation and fitness scripts and used populations. How objective parameters for the measurement can be found? The best evaluator would be a human listener with corresponding musical knowledge. The measurement results show average fitness of the populations after GA passed given count of generations. Every measurement was repeated six times.

In the graphs in Figures 4–7 and 9 axis x shows generation (0 means initial population), axis y shows the mutation operator weight, and axis z shows the acquired fitness.

6.1 Operator Bass-Teamwork

We will measure the influence of the Bass-teamwork operator upon the population fitness. The fitness script defines that we want a result with crash cymbal and bass drum with the same rhythm as the HAI. See HAI phrase in Figure 3.

⁴ This phrase is a part of a MIDI track.

⁵ This stroke is generated independently of GA.

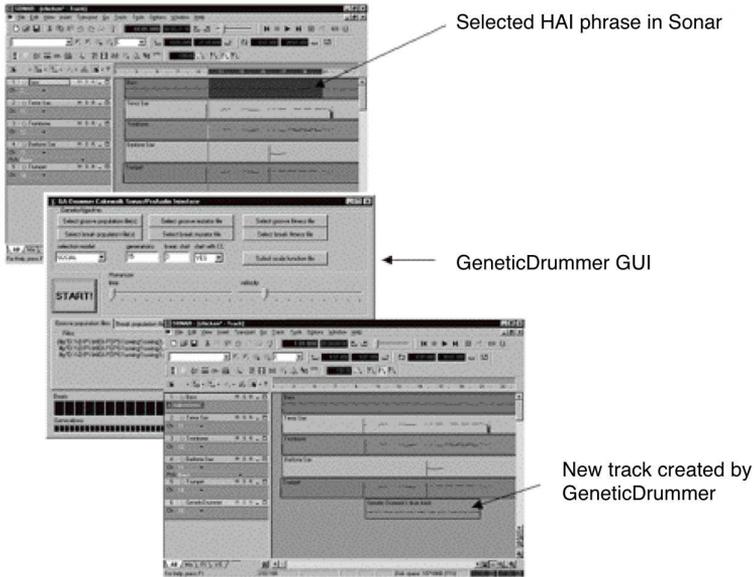


Fig. 2. User interface of GeneticDrummer and cakewalk sonar

```
(setq fitness-script
  '(list
    (list 60 'fitness-rythm-conformity 'notes-bass music)
    (list 60 'fitness-rythm-conformity 'notes-cc-1 music)
    (list -20 'fitness-rythm-unconformity 'notes-bass music)
    (list -20 'fitness-rythm-unconformity 'notes-cc-1 music)
    (list 22 'fitness-instrument-quantity 'notes-snare 0)
    (list 22 'fitness-instrument-quantity 'notes-tt-10 0)
    (list 22 'fitness-instrument-quantity 'notes-tt-12 0)
    (list 22 'fitness-instrument-quantity 'notes-tt-13 0)
  ))
```

Parameter w specifies mutation influence (mutation probability):

```
(setq mutation-script
  '(list
    (list 'notes-cc-1 'mutation-bass-teamwork <w> music)
    (list 'notes-bass 'mutation-bass-teamwork <w> music)
  ))
```


	1	2	3
0	100.8	101.3	82.7
1	137.3	150.6	100.3
2	167.3	163.5	117.1
3	178.8	175.8	130.9
4	179.9	177.8	145.2

Table 8. Bass-teamwork operator – mutation probability $w = 0.25$

	1	2	3
0	100.8	101.3	82.7
1	166.3	169.0	139.3
2	179.3	185.5	161.1
3	189.8	187.8	179.9
4	195.9	199.8	181.1

Table 9. Bass-teamwork operator – mutation probability $w = 0.6$

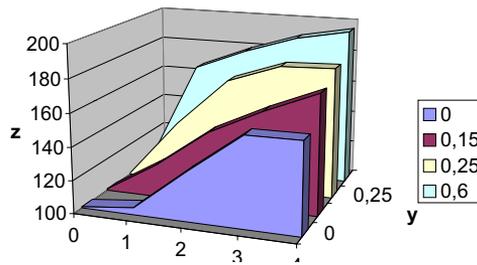


Fig. 4. Bass-teamwork operator – measure 1

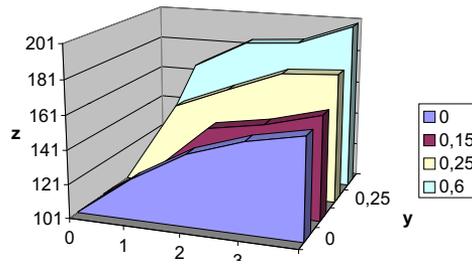


Fig. 5. Bass-teamwork operator – measure 2

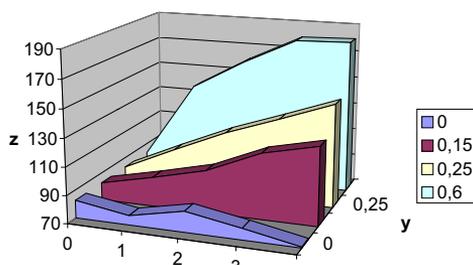


Fig. 6. Bass-teamwork operator – measure 3

```

))
(setq mutation-script
  '(list
    (list 'notes-snare 'mutation-syncopize <w> music)
  ))

```

It can be seen from Table 10 and Figure 7 that the population quality cannot be improved without mutation operator because the initial population is rhythmically straight; thus, this task could be solved by help of crossover operator. We will achieve enough fitness with a higher influence of the Syncopize operator only.

	0.00	0.15	0.30	0.60
0	1.0	1.0	1.0	1.0
1	1.0	1.2	2.4	2.0
2	1.0	1.6	3.9	3.8
3	1.0	2.2	4.5	6.0
4	1.0	2.6	4.7	7.3

Table 10. The influence of the Syncopize operator

6.3 Selectivity of GA

We will measure the ability of a population to imitate the dynamic structure (velocity changes) of the HAI (see Figure 8). GA will not use mutation operator at all. The first population contains twelve multichromosomes – the real practices for human drummers taken from [1]. Average quality of the population is 2.41, so it can be considered as rather appropriate to perform the mentioned task. The second population is designed large and rhythmically inconsistent and it is composed of 68 multichromosomes. Inconsistency is based on the fact that 80 % of multichromosomes are fully improper for the task specified above. Average fitness of a population is 1.41.

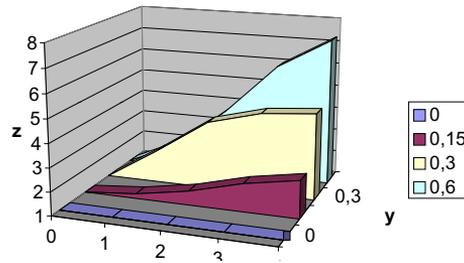


Fig. 7. Syncopize operator – graph

The experiment result can be seen in Figure 9. The x , y and z axes show quality, generation and (first or second) generations, respectively. It can be seen that GA is selective enough. When we used the second (improper) population we could see that there is a slower growth of the fitness than in the case of the first population. After approximately eight generations there is a comparable result to the first, “better” population. After approximately fourteen generations the second population achieves higher average fitness values than the first population. GA with the first population stops the fitness increase after approximately 18.7 generations and with the second population the fitness increase stops after approximately 38.25 generations.



Fig. 8. HAI-phrase

```
(setq fitness-script
  '(list
    (list 10 'fitness-dynamic-conformity 'notes-snare music-dynamics)
    (list 10 'fitness-dynamic-conformity 'notes-tt-10 music-dynamics)
    (list 10 'fitness-dynamic-conformity 'notes-tt-12 music-dynamics)
    (list 10 'fitness-dynamic-conformity 'notes-tt-13 music-dynamics)
    (list 10 'fitness-dynamic-conformity 'notes-ft-16 music-dynamics)
  ))
```

7 CONCLUSION

The described GA based system produces more realistic results than other mentioned systems. Rhythmic accompaniment generating is more independent from the user and can be considered as semi-automatic. Usage of higher-level musical structures (elimination of local constraints) can probably produce even better results. After all,

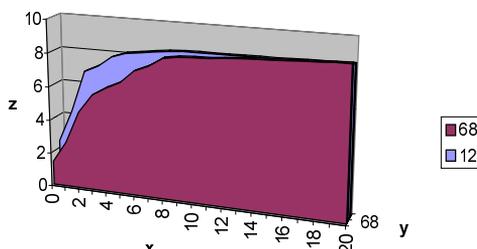


Fig. 9. Selectivity of GA

the local constraints allow generating of quite complicated and sophisticated results. On the other hand, the local constraints potentially allow (when a population is not rhythmically consistent) generating of two neighbouring beats which are mutually inconsistent. The configuration style of the fitness and mutation operators proved that this is achievable for a musically not so much erudite person. The performed experiments have revealed that at least four generations of GA must be passed to achieve appropriate results.

Please note that accompaniment generating is nondeterministic so that we get different (but still criteria satisfying) results if we run the generating process again with the same parameter values.

Visit <http://phoenix.inf.upol.cz/~dostal/evm.html> for some results of the GeneticDrummer system exported to MP3 format (for the piece Sockshop both initial groove and break population transferred to MP3 format are used. Each measure in the MP3 file represents one multichromosome of a population). The HAI phrases were manually selected by the author.

Acknowledgement

This paper is based on author's master thesis [3] supervised by Jozef Kelemen (Silesian University Opava, Czech Republic).

REFERENCES

- [1] BARTZ, J.: *Poradnik Perkusisty Amatora*. COMUK, 1986 (in Polish).
- [2] BILES, J.: *GenJam – A Genetic Algorithm for Generating Jazz Solos*. In *Proceedings of the 1994 International Computer Music Conference, ICMA*, San Francisco, 1994.
- [3] DOSTÁL, M.: *Genetické Algoritmy a Rytmus*. Master thesis, Silesian University, 2002 (in Czech).

- [4] GOLDBERG, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing, 1989.
- [5] HOROWITZ, D.: Generating Rhythms with Genetic Algorithms. In Proceedings of the 1994 International Computer Music Conference, ICMA, San Francisco, 1994.
- [6] JACOB, B.: Composing with Genetic Algorithms. In: Proceedings of the 1995 International Computer Music Conference, Banff Alberta, September 1995.
- [7] MANNING, P.: Electronic and Computer Music. Clarendon Press 1994.
- [8] MCCORDUCK, P.: Aaron's Code. Freeman Publishing 1990.
- [9] TANGUIANE, A. S.: Artificial Perception and Music Recognition. Springer-Verlag 1993.
- [10] TOKUI, I.—IBA, H.: Music Composition with Interactive Evolutionary Computation. In: Proceedings of 3rd International Conference on Generative Art (GA2000). 2000.
- [11] YO, CH.: Computer generated music composition. Master thesis, Massachusetts Institute of Technology 1996.



Martin DOSTÁL received his Ph.D. degree with the theme “A Functional Approach to Automatic Programming” in 2005. Currently he works as assistant professor at the Department of Computer Science of Palacký University in Olomouc. His main fields of interest are automatic programming, evolutionary computing and functional programming.