A Comparison of Genetic Operations to Produce Musical Structures

Project Report

Submitted by:

Damian A. Padrón Abrante

At the

Institute for Electronic Music and Acoustics (IEM)
Graz University of Music and Dramatic Arts
A-8010 Graz, Austria

In SS 2005

Supervisor: Mag. Gerhard Nierhaus

Abstract

My work for this project concerns genetic algorithms in the area of algorithmic composition.
Starting with a historical overview, a profound introduction into the principles of these algorithms is given, followed by a detailed description of important research in that field.
The main focus of this work is to compare the effectiveness of different genetic operations by reconstructing already composed musical material.

Contents

1. Evolutionary Computation

1.1 Introduction

The field of Evolutionary Computation (short: EC) can be divided in four different groups, although all these parts are inspired by the principles of evolution enunciated by Charles Darwin[1]. In his book, Darwin propounded the theory that all species are originated by means of the process of natural evolution and introduced the elemental concepts of genotype, phenotype, expression, selection and reproduction with variations.

The genotype is the genetic encoded information for the formation of an individual. The phenotype is the individual itself. Expression is a process by which a phenotype is produced from a genotype. The process of selection depends on the kind of procedure by which the fitness[2] of phenotypes is stipulated. Reproduction is the process by which a new genotype is generated from an existing genotype. The reproduction with variations leads the evolution of a population towards a higher level of fitness. These variations are called mutation and crossover. The mutation mildly modifies a single gene[3] to produce a new gene. Crossover exchanges information between two genes to produce two new genes.

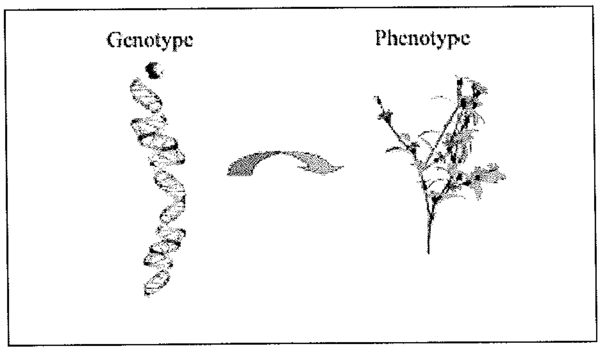


FIGURE [1]: genotype and phenotype [González 2003, pp. 5]

[1] [Darwin 1859]
[2] Numerical index that expresses the ability of an organism to reproduce and survive.
[3] Section of a chromosome which transmits a peculiar hereditary characteristic.

EC is made up of four fields: Evolutionary Programming (short: EP), Evolution Strategies (short ES), Genetic Programming (short: GP) and Genetic Algorithms (short GA).

1.2 Evolutionary Programming

EP was proposed by Lawrence Fogel[4] while examining the use of simulated evolution as a way of developing artificial intelligence (short: AI).
Evolutionary Programming stresses phenotypic adaption. They emphasize the link between the parent chromosomes and their descendants. Every chromosome identifies the behaviour of a Finite State Machine (short: FSM). An FSM is a [...] "machine defined in terms of a finite alphabet of possible input symbols, a finite alphabet of possible output symbols, and some finite number of possible different internal states" [...] [Fogel 1966, p.12]. Each state is showed as a node in a network and the network entail indicates the input / output state transitions.

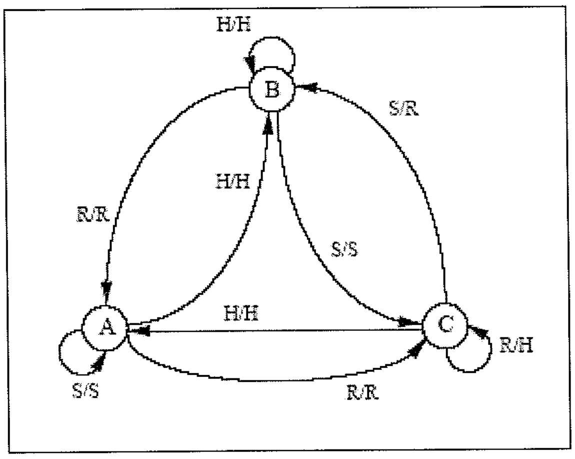


FIGURE [2]: Example of a weather predicting Finite State Machine (FSM). The nodes indicate states and the links indicate input / output state transitions. [Collins 2000, pp. 5]

Steps for a basic algorithm with EP:

- Generate an initial population randomly.
- Apply mutation.
- Calculate the fitness of every child, and afterwards selection is carried out by means of tournament.

For example, a finite automaton can be changed through mutation to identify some inputs.

[4][Fogel 1966]

Mutation can change an output symbol, change a transition, add a state, delete a state and change the initial state.

Actual State	A	A	B	B	C	C
Input symbol	0	1	0	1	0	1
Next state	B	C	B	C	B	C
Output symbol	N	N	Y	N	N	Y

FIGURE [3]

1.3 Evolution Strategies

The concept of ES was developed by Ingo Rechenberg[5] and Hans Paul Schwefel[6]. They were working on the quest of the optimum forms of bodies in a fluid. In its early stages, ES were not concrete algorithms to be used with computers but a way to discover optimal parameters in laboratory experiments. The ES are focused on phenotypic transformation.

Hans Paul Schwefel worked on the first computer implementations based on ES[7]. Between 1977 and 1985 small works were done in this field. Nevertheless, during the 1990's, ES investigation was revived[8] due to financial support. In principle, these strategies were used to optimize hydro-dynamical complex problems, like minimizing the drag of a joint plate[9], or the optimization of structures[10]. These ES were developed by means of a series of discrete adaptations, such as mutations, to an experimental structure. After each adaptation, the new structure (offspring) was compared and evaluated with respect to the previous structure (parents). The better of both, was chosen and utilized in the next cycle. As selection in this cycle is made from one parent and one offspring, the algorithm is known as a (1 + 1)-ES.

5 [Rechenberg 1973]
6 [Schwefel 1977]
7 [Schwefel 1975]
8 [Schwefel 1995]
9 [Rechenberg 1965]
10 [Schwefel 1968]

- One parent generates only one offspring. The offspring was kept only if it was better than his parent.
- A new individual is generated introducing Gaussian Noise $X_{t+1} = X_t + N(0,\sigma)$, where N is an independent number Gaussian array, with zero average, and standard deviation 'σ'.

For example, supposing the next function is to be optimized: $f(x_1;x_2) = 100(X_1^2 - X_2)^2$, where $-2048 \leq x_1, x_2 \leq 2048$.

At the same time, the next individual randomly generated (-1,1) and the next mutations are given:

$$x_1^{t+1} = x_1^t + N(0,1.0) = -1.0 + 0.61 = -0.39$$
$$x_2^{t+1} = x_2^t + N(0,1.0) = 1.0 + 0.57 = 1.57$$
$$f(x_t) = f(-1,1) = 4$$
$$f(x_{t+1}) = f(-0.39,1.57) = 201.416$$

Rechenberg introduced the concept of population in which M parents generate one offspring[11]. Schwefel introduced the utilization of multiple offspring, where it is possible to choose the better "M" offspring or the M better individuals, considering the parents and the offspring[12].

Furthermore, Rechenberg formulated one rule to adjust the standard deviation during the evolutionary process[13]. Its name is "1/5 success rule", and it says: 1/5 must be the rate between successful mutations and global mutations.

1.4 Genetic Programming

John Koza is the author of the genetic programming domain. He began writing about this area in 1990, and demonstrated that GP operated on an extended number of AI problems and published varied papers[14].

---

11 [Rechenberg 1973]
12 [Schwefel 1977]
13 [Rechenberg 1973]

GP offers a solution through computer programs based on methods of natural selection. In fact, GP is a modification of GA to develop computer programs, like, for instance, simple bit-strings.

In accordance with Koza[14], a definition of genetic programming can be the evolution of *tree-structures*. This is a rigorous definition. There is other definition given in the work of Banzhaf[15], which affirms the following:

1. The term GP includes all systems which constitute or include explicit references to programs or to programming language expressions.
2. The definition of genetic programming contains all ways of representing programs.
3. Algorithms which are not primarily programs, like artificial neural networks, shouldn't be excluded from this definition.
4. This definition is not limited to the use of crossover; all systems which use a population of programs or algorithms for the good of search are incorporated.

Before employing genetic programming in a problem, we must take the next steps:

1. Define the set of terminals.
2. Choose the set of primitive functions.
3. Define the fitness function.
4. Determine the parameters to control the run.
5. Select a method for developing a result of the run.

Any computer program is a sequence of operations[16] applied to values[17], but distinct programming languages allow including different kinds of operations and statements, and having distinct syntactic restrictions. The principal language for genetic programming is LISP, because it enables a computer program to be manipulated as data and the recently created data to be executed as a program.

[14] [Koza 1992]
[15] [Banzhaf 1998]
[16] Functions
[17] Arguments

LISP has a symbol-oriented structure. Its basic data structures are atoms[18] and lists[19]. The more used representation is the program tree.

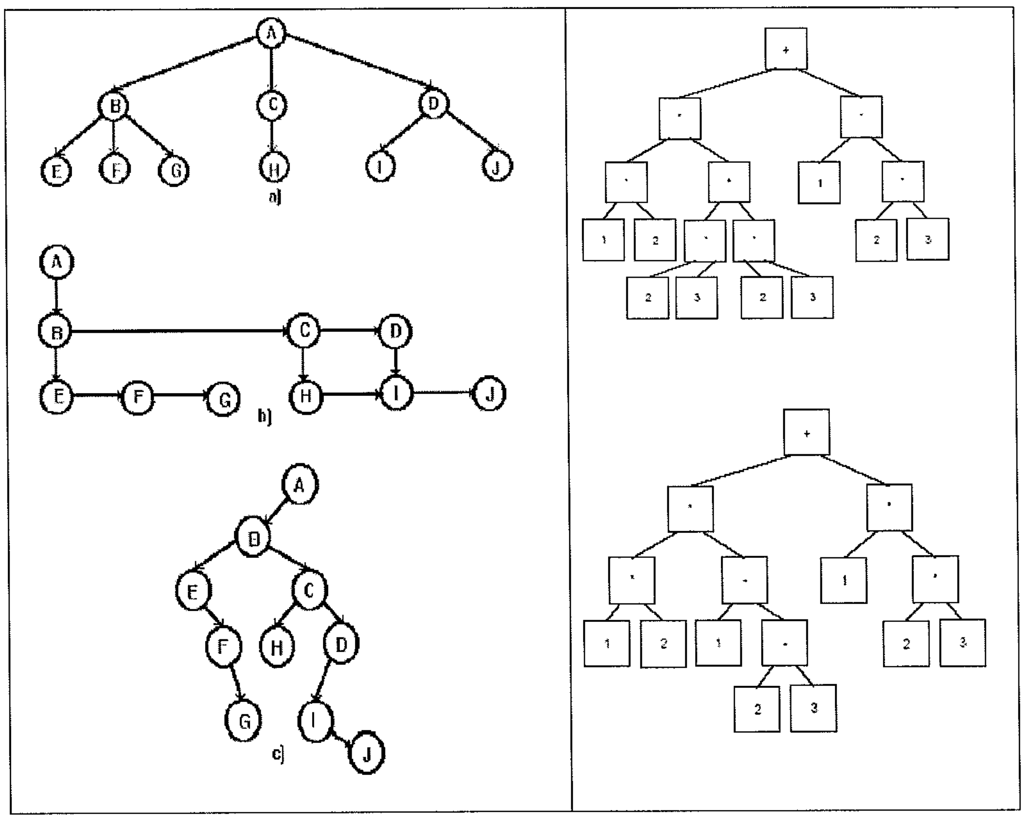


FIGURE [4]: Different kinds of tree representations, [Torres 2005, pp. 17]; [Velikonja 2003, pp. 12-13].

1.4.1 Example of LISP Structure

If next list is given:

(-(* A B) C)

It calls for the application of the subtraction function (-) to two arguments, particularly the list (* A B) and the atom C. First, the LISP language, applies the multiplication function (*) to the atoms A and B.

Once the list (* A B) is evaluated, LISP applies the subtraction function (-) to the two arguments, and therefore evaluates the entire list: (-(* A B) C).

[18] Smallest indivisible element of the LISP syntax

[19] Object composed of atoms and/or other lists

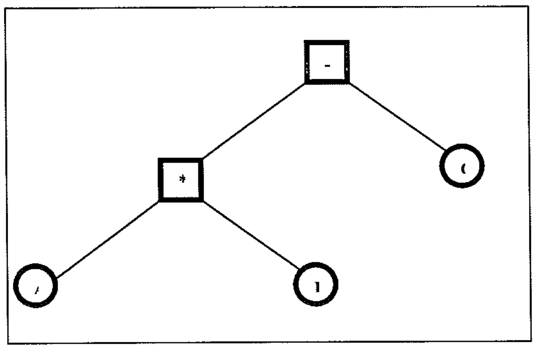


FIGURE [5]

1.4.2 Example of 11-Multiplexer Problem with LISP Structures

This a problem described by Koza[20]. The problem consists of creating a Boolean function (or circuit) which implements a Boolean 11-multiplexer. The multiplexer possesses 11 inputs and one output.

The inputs a0, a1 and a2, are the address lines and describe the binary representation of an integer number between 0 and 7. This integer chooses which of the 7 inputs identified as d0, d1, d2, d3, d4, d5, d6 and d7 is selected. The correct input is located in the line indicated by the address line. In the figure [6], the address line selects the integer 6, in such a manner that the output should be the input on line d6, which is 1.

Then, there are 2^{11} possible inputs to this function, and all these inputs can be tested. The raw fitness is the number of cases for which the output is correct, and the standardized fitness is 2048 less the raw function.

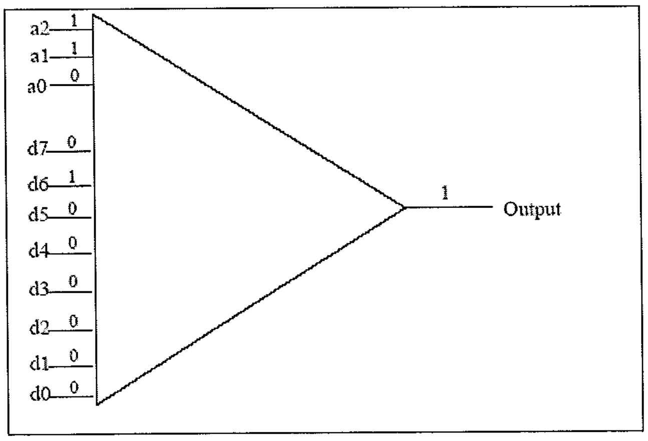


FIGURE [6]: 11-multiplexer with an input of 11001000000 and output of 1. [Koza 1992, pp. 171]

[20][Koza 1992]

The inputs to the function are the terminals. The non-terminals are AND, OR, NOT, IF. Koza used a population size of 4000.
Next, the worst-of-generation individual for generation 0 is shown:

```
(OR (NOT A1) (NOT (IF (AND A2 A0) D7 D3)))
```

This individual had a standardized fitness of 1280 (the raw fitness is only of 768), but its functioning was really bad for all 2048 combinations of the 11 terminals. 23 individuals in this initial population bound up with the highest score of 1280 matches on generation 0. One of these high scoring individuals was the S-expression:

```
(IF A0 D1 D2).
```

Afterwards, in the generation 1, the raw fitness of the best-of-generation individual went up to 1408. Only one in the population achieved this high score, namely

```
(IF A0 (IF A2 D7 D3) D0).
```

The generations continued, and a solution was discovered in generation 9. This solution is:

```
(IF A0 (IF A2 (IF A1 D7 (IF A0 D5 D0))
     (IF A0 (IF A1 (IF A2 D7 D3) D1) D0))
    (IF A2 (IF A1 D6 D4)
         (IF A2 D4 (IF A1 D2 (IF A2 D7 D0)))))
```

Koza justifies crossover by describing the crossover occurrence that produced the best individual:
"[...] Even though neither parent is perfect, these two imperfect parents contain complementary, co-adapted portions which, when mated, produce a 100%-correct offspring individual. [...]" [Koza 1992, p.185]

1.5 Genetic Algorithm

Genetic algorithms were invented by John Holland[21], and they are techniques based on genetics and the mechanics of natural selection, including an exchange of structured and randomized information bringing about survival of the fittest within a population of string structures. The main Holland's innovation was the introduction of a population-based algorithm with inversion, crossover and mutation. Besides, Holland was the first person to try to present computational evolution as a solid theoretical basis.

These are the central dogmas of genetic algorithms by Holland[21]:

- Schema processing
- Schema theorem
- Implicit parallelism
- Building block hypothesis
- K-armed bandit analogy

• Schema Processing

There are high dimensional search spaces, like binary strings of length "l". The GA use predisposed sampling to search high dimensional spaces[22].

The schemas take regularities in the search space:

0	1	**0**	0	**1**	**1**
1	0	0	1	1	1
x	x	x	x	**1**	**1**
x	x	**0**	x	x	x
x	x	**0**	x	x	**1**
x	x	**0**	x	**1**	**1**

FIGURE [7]

In accordance with Schema Theorem, the crossover and reproduction guarantee exponentially increasing samples of the studied best schemas. The order of a schema is

[21][Holland 1975]

[22] Independent sampling or selection search towards high fitness areas. [23] Number of defined bits.

O(s)[23] while D(s) is the distance between the bits of the extremes.

• Schema Theorem

Example of a schema theorem:

S is a schema in a population at time t. Then, N(s,t) will be the number of cases of s at time t. The expected number of offspring(x) will be:

$$\text{offspring}(x) = \frac{F(x)}{\overline{F}(t)}$$

Ignoring mutation and crossover, this is the expected N(s, t +1)

$$N(s,t+1) = \frac{\hat{\mu}(s,t)}{\overline{F}(t)} \times N(s,t).$$

Mutation and crossover handled as loss term:

$$N(s,t+1) = \frac{\hat{\mu}(s,t)}{\overline{F}(t)} \times N(s,t)\left(1 - p_c \frac{D(s)}{l-1}\right)\left[(1-p_m)^{O(s)}\right]$$

In a population consisting of N individuals and each individual is L bits long, in one generation, there will be a number of sampled schemas by the population between the values: 2^L and $Nx2^L$

• Implicit Parallelism

In the implicit parallelism, one individual samples many schemas simultaneously, and this makes GA a very effective optimization algorithm. Its greatest advantage is the ability to find approaching solutions to different combining problems.

• Building Block Hypothesis

1. At first, GA locates biases in low order schemas: GA gets fine estimates of schema average fitness by sampling strings.
2. The information from low order schemas is combined through crossover, and
3. GA locates biases in high order schemas, casually converging on the fittest region of the space.

• K-armed Bandit Analogy

The two armed bandit is based on adaptive control and statistical decision theory.

Suppose N coins are given to play a slot machine with two arms (A_1 and A_2). The arms have variances s_1 and s_2, and have a mean payoff per attempt which rates between m_1 and m_2.

The payoff procedures of the two arms are static and totally independent of each other. The wagerer does not know these payoffs, and can value them only by playing coins on distinct arms.

If the player wants to maximize total payoff, he has to find the best strategy, that is, maximize payoffs during N attempts (on line payoff) or determine which arm possesses the highest payoff rate (off line payoff). The optimum strategy is to exponentially increase the sampling rate of the studied best arm, as more samples are gathered.

In 1975, De Jong finalized his dissertation[24]. De Jong's study combined Holland's theory and his own careful computational experiments and applied GA to function optimization. This work is concerned with the analysis and design of adaptive systems, especially in the field of adaptive computer software. The central characteristic of the evaluation process is fastness: the capacity of an adaptive system to quickly reply to its environment in an ample range of situations. A new class of genetic adaptive systems is established for analysis and evaluation. These artificial genetic systems[25] produce adaptive responses by simulating the information obtained in natural systems through the mechanism of evolution.

In 1989, David Goldberg[26] finished his most important book[27]. This text presented the GA as an approach to resolve search problems of several kinds. It presented the genetic algorithm as a problem solving tool. In accordance with Goldberg, the GAs are "[...] search algorithm with some of the innovative flair of human search [...]" [Goldberg 1989]. He collates conventional search methods with GA, concluding: "[...] while our discussion has been no exhaustive examination of the myriad methods of traditional optimisation, we are

[24] [De Jong 1975].
[25] Called reproductive plans
[26] A John Holland's early student.
[27] [Goldberg 1989].

left with a somewhat unsettling conclusion: conventional search methods are not robust. [...]" [Goldberg 1989, p.5]

His book starts with an introduction to genetic algorithms, in which a simple genetic algorithm, simulations by hand, which are different from traditional methods, and some problems are presented. Afterwards, the fundamental concepts like fundamental theorem, Schema Processing, the two-armed and k-armed bandit problem or building block hypothesis are explained. All these concepts have been explained in the paragraph 1.5.

Next, he explains how to implement a genetic algorithm, starting with the data structures like the data type declarations. Next, he implements a reproduction (by selection), a single-point crossover, a single bit point mutation, a generation of a new population, a decoding (he decodes a binary string like a single), and a main program for this simple GA, all in Pascal code. He offers two principles for choosing a GA coding:

- <u>Principle of meaningful building blocks</u> [Goldberg 1989, p. 80]:

"[...] The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.[...]"

- <u>Principle of minimal alphabets:</u>

"[...] The user should select the smallest alphabet that permits a natural expression of the problem.[...]"

He gives different coding routines to use in GA, and explains the discretization of parameters and the constraints. Besides, he enumerates some applications of genetic algorithms like biological cell simulation, function optimization, optimization of pipelines systems, etc....

In this point, Goldberg, deepening into other fields, introduces a series of more difficult concepts related to advanced operators and techniques in genetic search (dominance, diploidy, abeyance, micro-operators, parallel processors, etc) and other terms related to genetics based on machine learning (classifier system, rule and message system, development of the first classifier system and current applications).

The book ends with several interesting appendixes: a review of combinatorics and elementary probability; an introduction to the Pascal Language (simples codes, functions, procedures) and a lot of distinct genetic algorithms in Pascal, like a simple genetic algorithm, a simple classifier system or partition coefficient transforms for problem-coding analysis.

```
function select (popsize:integer; sumfitness:real;
var pop:population): integer;
var rand,partsum: real
j:integer;

begin
partsum:= 0.0;
j:= 0;
rand := random * sumfitness;
repeat
j:=j+1;
partsum:=partsum + pop[j].fitness;
until (partsum >= rand) or (j=popsize);
select:=j;
end;
```

```
function mutation(alleleval:allele; pmutation:real;
var nmutation:integer):allele;

var mutate:boolean;
begin
mutate:=flip(pmutation);
if mutate then begin
nmutation:=nmutation+1;
mutation:= not alleleval;
end else
mutation:=alleleval;
end;
```

FIGURE [8a]: Reproduction (select).[Goldberg 1989, pp.63]. **FIGURE [8b]:** Single bit, point mutation.[Goldberg 1989, pp.65]

```
procedure crossover(var parent1,parent2, child1,child2:chromosome;
var lchrom,ncross,nmutation,jcross:integer;
var pcross,pmutation:real);
var j:integer;
begin
if flip(pcross) then begin
jcross:= rnd(1,lchrom - 1);
ncross:= ncross + 1;
end else
jcross:=lchrom;

for j:=1 to jcross do begin
child1[j]:= mutation(parent1[j],pmutation,nmutation);
child2[j]:= mutation(parent2[j],pmutation,nmutation);
end;

if jcross <> lchrom then
for j:= jcross+1 to lchrom do begin
child1[j] := mutation(parent2[j],pmutation,nmutation);
child2[j] := mutation(parent1[j],pmutation,nmutation);
end;
end;
```

FIGURE [9]: Single-point crossover. [Goldberg 1989, pp. 64]

1.5.1 Principal Elements of Genetic Algorithms

The might of genetic algorithms is situated in its fitness function[28] and genetic encoding[29]. GAs differ from classic algorithms in four ways[30]:

1. GAs usually work with a coding of the parameter set, not with the parameters themselves.
2. GAs search on a population of points, not on a single point.
3. GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules, not deterministic rules.

In a general manner, a genetic algorithm consists of the next steps[31]:

1. Create an initial population of solutions
2. Evaluate each solution and assign it a fitness value
3. Select "parents" of the next generation of solutions based on these fitness values
4. Generate "children" from these parents using crossover, cloning, and mutation.
5. Repeat steps 2-4 until finished.

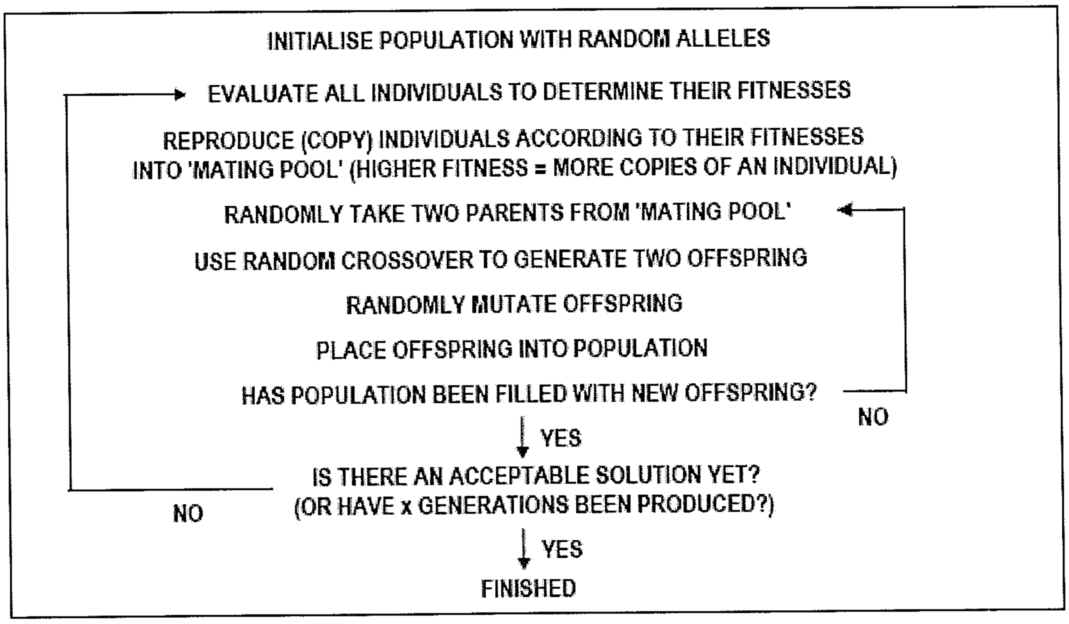

INITIALISE POPULATION WITH RANDOM ALLELES
EVALUATE ALL INDIVIDUALS TO DETERMINE THEIR FITNESSES
REPRODUCE (COPY) INDIVIDUALS ACCORDING TO THEIR FITNESSES INTO 'MATING POOL' (HIGHER FITNESS = MORE COPIES OF AN INDIVIDUAL)
RANDOMLY TAKE TWO PARENTS FROM 'MATING POOL'
USE RANDOM CROSSOVER TO GENERATE TWO OFFSPRING
RANDOMLY MUTATE OFFSPRING
PLACE OFFSPRING INTO POPULATION
HAS POPULATION BEEN FILLED WITH NEW OFFSPRING?
NO
YES
IS THERE AN ACCEPTABLE SOLUTION YET? (OR HAVE x GENERATIONS BEEN PRODUCED?)
NO
YES
FINISHED

FIGURE [10]: The simple genetic algorithm [Bentley 1997, pp. 3]

28 Provides the set of possible solutions and the reproduction operators for this set.
29 It shows if the genetic algorithm is directing in the right guidance.
30[Goldberg 1989]
31 [Mc Auley 2004]

1.5.2 Definitions

- **Child:** a gene[32] that is the result of a reproduction operator

- **Parent:** incoming gene to a reproduction operator.

- **Genotype:** representation of a gene, like a bit-string or a list of values. Other definition would be: genetic encoded information for the conception of an individual.

- **Phenotype:** solution corresponding to a particular gene; [...] "the individual itself, or the form that results from the developmental rules and the genotypes" [...] [33]

- **Decode:** transformation of a genotype into the respective phenotype.

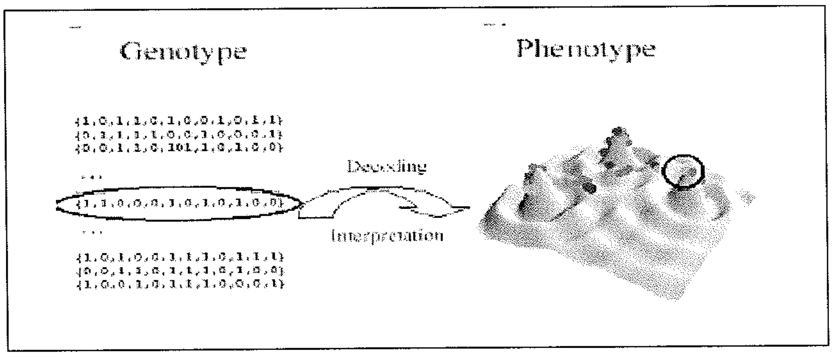


FIGURE [11]: Decoding process [González 2003, pp. 10]

- **Fitness:** is a numerical indicator expressing the capacity of an organism to reproduce and survive.

- **Fitness function:** supplies a measure of the quality of a chromosome.

- **Encode:** transform a phenotype to the respective genotype. A common form of encoding is the bit string.

- **Population:** is composed of encoded representations of possible solutions. The population develops by means of reproduction operators.

- **Selection Probability:** Probability of a gene to be selected for reproduction. It's based on the rank of its fitness into the population.

- **Competition:** In accordance with the selection probability, gives advantage to genes with

---

[32] The basic structure manipulated by the GA which depicts a particular solution to the problem that we want to solve.
[33] [Moroni 2000].

elevated fitness versus genes with inferior fitness.

- **Generational update:** Replacing the complete population, but before recalculating fitness and permitting new genes to reproduce.
- **Steady state update:** Replacing a small number of genes, recalculating fitness and permitting new genes to reproduce.
- **Reproduction:** Is the process by which new genes are generated from old genes.
- **Reproduction operator:** A genetic operator takes a small group of genes (generally 1 or 2) and generates an actualized group of genes. There are two classes of genetic operators:

- **Crossover:** operator that interchanges bits between two genes. There are, principally, three types of crossover: 1-point crossover, 2-point crossover and the uniform crossover.

1	0	1	1	0	1	1	0	→ One Point …	1	0	1	1	0	0	0	1
1	0	0	1	1	0	0	1	Crossover →	1	0	0	1	1	1	1	0

FIGURE [12a]

1	0	1	1	0	1	1	0	→Two Point …	1	0	0	1	1	1	1	0
1	0	0	1	1	0	0	1	Crossover →	1	0	1	1	0	0	0	1

FIGURE [12b]

1	0	1	1	0	1	1	0	→Uniform …	1	0	0	1	0	0	1	0
1	0	0	1	1	0	0	1	Crossover →	1	0	1	1	1	1	0	1

FIGURE [12c]

- **Mutation:** operator that modifies a single gene lightly to produce a new gene. In other words, mutation takes a chromosome and changes part of it in a random way.

1	0	1	1	0	1	1	0	→Mutation →	1	0	0	1	0	1	1	0
1	0	0	1	1	0	0	1		1	0	0	1	1	0	1	1

FIGURE [13]

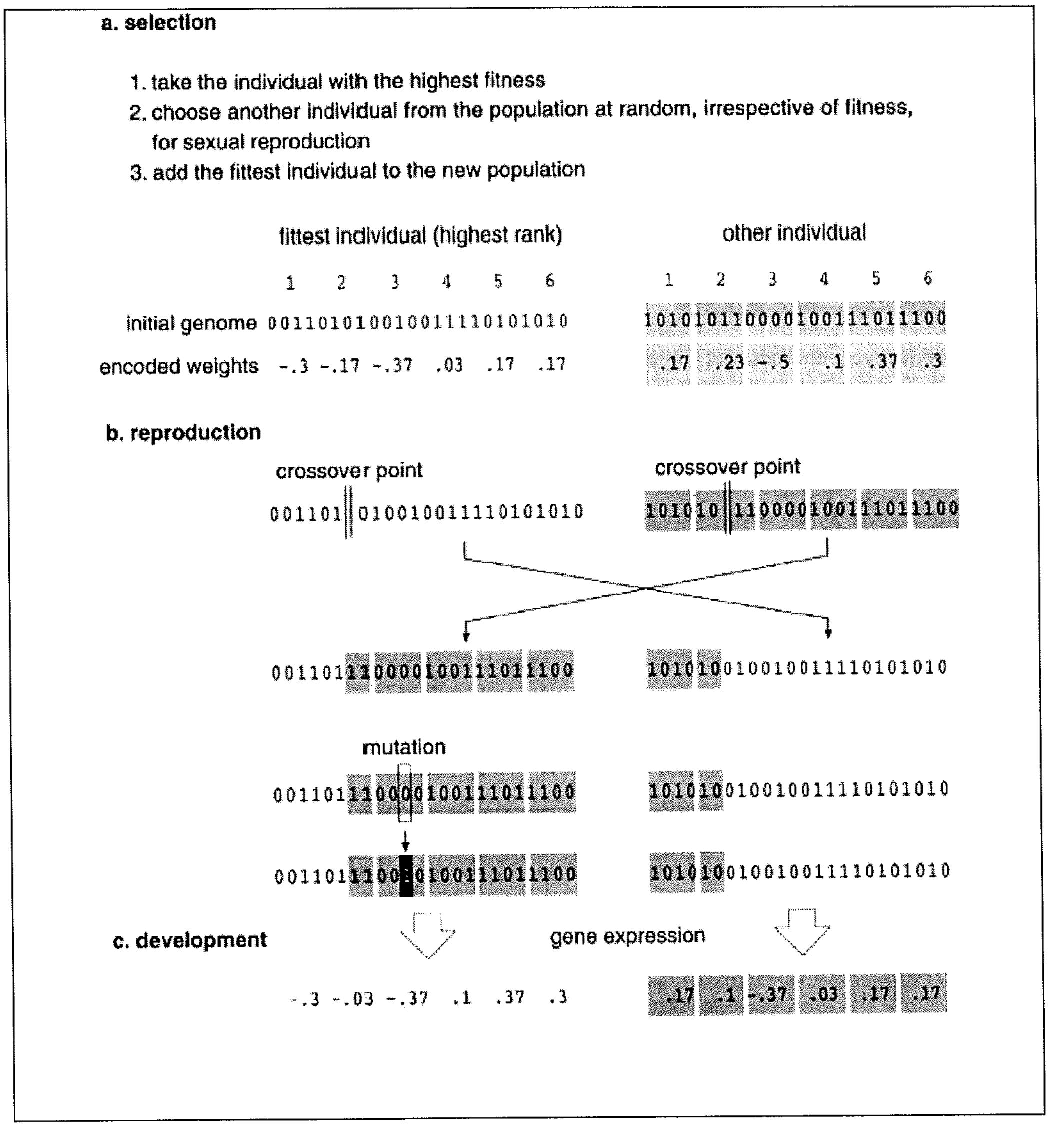


FIGURE [14]: Example of selection and reproduction [Pfeifer 2003, pp.9]

a) Selection: After their final fitness values have been determined, individuals are selected for reproduction.

b) Reproduction: The crossover point is chosen at random. The entire population is subjected to a small mutation.

c) Development: After reproduction, the new genome is expressed to become the new individual.

2. Evolutionary Computation in Musical Composition

In this paragraph, some of the important works about GAs applied to the musical composition are presented in chronological order.

Horner carries out the first work aimed at musical composition[34], but in a minimalist way. He uses the genetic algorithms for thematic bridging among simple melodies. In the work of Takala[35] it is possible to find the use of physically-based models and GA for functional composition of sound signals synchronized to animated motion. He represents sound signals as general functional compositions, whose name is timbre trees[36].

Jacob applies the GA to the problem of composing music[37]. He presents a system called "The Variations", and describes different examples and Biles describes the use of an IGA[38] into his "GenJam" system to generate jazz solos on an input chord progression[39].

Damon Horowitz presents a genetic algorithm for the generation of rhythmical textures and percussive material[40]. In McIntyre's system a GA generates a four-part Baroque harmonisation of a melody given by the user[41]. He used only the C major scale.

In the year 1995, Ralley uses a GA system to generate tunes from an input melody, given by the user. These melodies are restricted to 12 notes[42].

Later, Biles presents another version of "GenJam"[43], in which a group of people criticizes the solos produced by the system, and the search of a neural network fitness function for the implementation of a musical interactive GA[44].

Werner shows in his work[45] a co-evolutionary forming approach to signal design to probe signal patterns in birds. In the same year, Papadopoulos and Wiggins use a symbolic GA to search in a space of potential solutions[46]. The objective is to get a system that generates jazz melodies based on an input chord progression.

---

34 [Horner 1991]
35 [Takala 1993]
36 These are LISP expressions, internally implemented as C++ structures
37 [Jacob 1994]
38 Interactive Genetic Algorithm.
39 [Biles 1994]
40 [Horowitz 1994].
41 [Mc Intyre 1994]
42 [Ralley 1995].
43 [Biles 1995]
44 [Biles 1996].
45 [Werner 1997]
46 [Papadopoulos 1998]

In a new Biles' paper[47], a new version of the GenJam system is provided.
The Moroni's paper[48], describes the use of IGA in composition, centring on strategies used in the "Vox Populi" system to supply control of the fitness function in order to value harmonic and melodic features. Towsey describes several melodic features utilized as the basis for a GA fitness function and for different mutation procedures[49].
Gartland Jones'[50] first document shows a generative music system that uses a specific domain; knowledge abundant in GA, and other utilizations through the development of GAs use.
In Pigg's[51] document, an implementation of a genetic musician is created by means of two different genetic algorithms, having the goal of producing a piece of coherent music.
And, in the year 2003, Gartland Jones presents two papers. The first[52] describes the design and construction of another generative music system, a real-time composition system called "Music Blox". In the second[53], he investigates some aspects of using GAs for musical composition and their limitations, and presents the "Indago Sonus" system as one possible application.

2.1 Classification [54]

In a musical composition system, two roles can be distinguished: creator and critic (author and public). The different works have been organized in accordance with the critic, being the creator, in all the instances, represented by an EC system.

2.1.1 Interactive Systems

In these systems the critic is directly a user. In a system of this kind, the user aesthetically values each song or composition, guiding its evolution. The system takes into account this appreciation in the creation of the next compositions. In its easier form, these systems present the problem of temporal cost (or bottleneck) that implicates the human participation

[47] [Biles 1998]
[48] [Moroni 2000b]
[49] [Towsey 2001]
[50] [Gartland-Jones 2002]
[51] [Pigg 2002]
[52] [Gartland-Jones 2003a]
[53] [Gartland-Jones 2003b]
[54] Classification based on Santos' paper [Santos 2000].

[Biles 1994], [Papadopoulos 1999]. Besides, this problem can cause tiredness in the user. On the other hand, these methods have a high level of subjectivity.

Into this category, the works of Jacob[55] can be included. He invents "The Variations" system to implement his composition via interacting modules, and it was planned to reproduce very strictly the creative procedure that the author utilizes when composing music. In these works exist various layers. In the first place, an evolutionary critic is interactively adjusted through evaluation by the critic and the user of musical pieces. Immediately, the critic chooses, as input to other module, fragments produced via stochastic process or provided by the user. This second evolutionary module includes an adjustment done by the user and the evolution of each type of module is executed separately, by means of human user fitness feedback.
This process is similar to writing cannon. The system makes simple the music organization working at the level of motives. It's easier to produce structures in a piece working at a higher level than trying to work at the notes level. Next, the steps followed by Jacob for the construction of his algorithm[56] are explained:

1. Define a number of primary themes (motives) to be used in the composition.
2. Compose phrases by creating motives and adding them one by one to the phrase. At each step, judge the quality of the resultant phrase and remove the last motive if the combination is unsuitable.
3. Create motives by selecting at random from the primary themes and motives already in the phrase, and producing variations upon the selection.
4. Once there are a large number of phrases, join them together into larger frameworks.

In Biles'[57]work, the "GenJam Populi" system is presented. He tries to face the fitness bottleneck by patterning a system that enables multiple users to work in parallel to value the fitness of population members. This system generates a series of musical motives

[55] [Jacob 1994], [Jacob 1996]
[56] [Jacob 1996]
[57] [Biles 1995]

beginning with the evaluation of a user that occupies the censor role and a set of genetic operators adapted to the musical domain. Having as a base these motives, and starting from improvisations of Jazz solos interpreted by other user (who plays the interpreter role), it generates new solos.

The GenJam's interface utilizes easily attainable components to implement the following design. This system permits a single mentor to supply real-time feedback while "GenJam" extemporizes solos. There are various switches available, one for every mentor. Each box supplies a 'good' and a 'bad' switch. The switch boxes are connected to a microcontroller which maps the switch clicks to MIDI control change messages. This MIDI control sends them, by means of a standard MIDI cable, to the Yamaha MU-80 tone generator's MIDI IN station and, through the MU-80, to the host computer.

There is another train of works which not utilizes initial musical information. Various works of this class tackle the problem of the rhythmic composition. Horowitz presents in his work[58] different approximations to the rhythmical texture generation through GAs. The users listen to and value collection of rhythms that have already been developed through some generations of rule-based fitness evaluation. In the system, each generated piece can be evaluated and the wanted piece can be defined according to high level musical parameters (syncopation, density, beat repetition downbeat, etc...)

He gives us a group of constraining suppositions from which a big number of rhythms can be produced and his system utilizes an IGA to discover the user's criteria for differentiating between rhythms. The system evolves an increasingly precise model of the function that symbolizes the user's selection; the quality of all the generated rhythms is enhanced to fit the user's taste. The IGA are suitable to find a solution for this problem because they permit a user to perform a fitness function, that is, to select which rhythms he likes. And due to this reason, the user has not necessarily understanding of the parameters or details of this function.

Another work in the rhythmic domain is "Tribe"[59], which is a system inspired in more primitive music. This system permits to model music tribes. Every tribe musician is

[58] [Horowitz 1994]
[59] [Pazos 1999a], [Pazos 1999b], [Pazos 1999c]

associated to an instrument and he can only relate with other musicians that possess the same instrument. The user evaluates the whole piece.

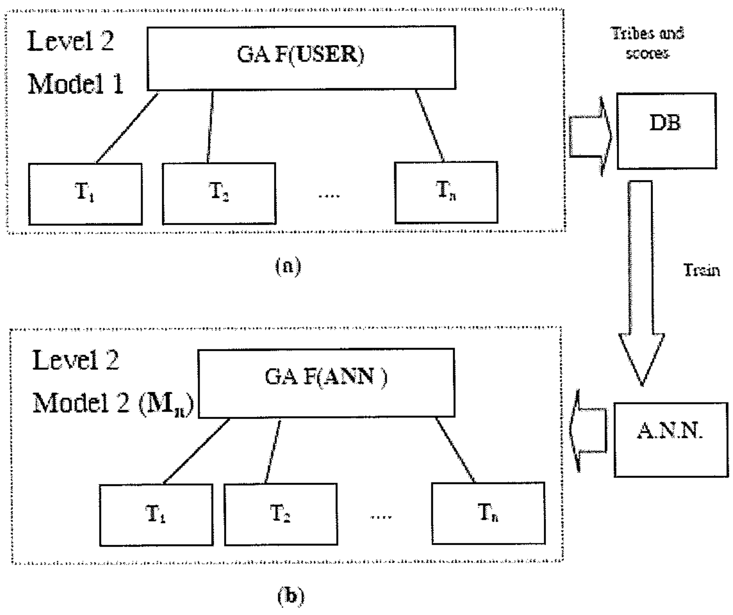


FIGURE [15]: Tribe system, relation between the interactive model (a) and the automatic model (b). [Pazos 1999c, pp 1]

The genetic algorithm selects the worst tribes as individuals to be removed. At the same time, it selects those tribes which will be used as parents. The system uses a random function for this selection so that each tribe has a probability proportional to its punctuation. The novel tribe is created by crossing the parent with every individual. Subsequently, mutation takes place in the produced individuals. The information added to the system (scores bestowed by the users, the several tribes created, etc) are registered in a database.

2.1.2 System Based on Artificial Neural Networks (ANN)

This subsystem is normally constituted by an ANN which is trained by theme songs. These theme songs are examples extracted from some musical style, concrete author or pieces created by interactive systems.

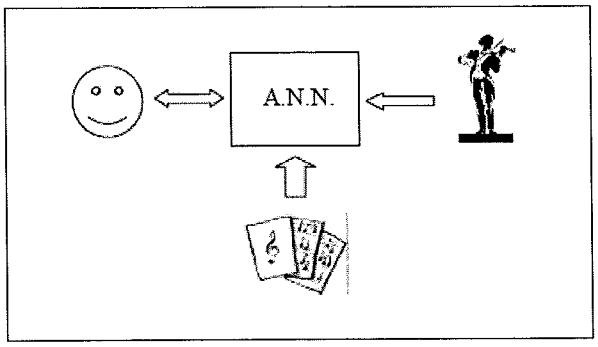


FIGURE [16]: The user introduces a succession of examples in order to train the ANN that will work like a critic of the evolutionary system's compositions. [Santos 2000, pp. 2]

In this kind of system we can include Burton's work[60]. In this work, he uses a network that classifies rhythmic songs or drum patterns from distinct styles of music (disco, funk, rock, fusion, etc...) originated from rhythm boxes. This ANN realizes a classification of rhythmic sequences adding new categories if the song doesn't enter in none of the existing class.

Other examples of this type are the Spector and Alpen's works[61], that incorporate genetic programming created by Koza[62]. In these systems, the individuals are functions that, from previous fragments, generate a new fragment. The music-making programs could catch examples of melodies from the case-base and alter them with a series of predetermined functions (invert, transpose, augment...) They use five critical functions from jazz improvisation rules to evaluate the response, and run their system with five Charlie Parker song pieces of four bars each one.

[60] [Burton 1998]
[61] [Spector 1995]
[62] [Koza 1992]

```
(+ (IF-LESS (IF-LESS 16 14 35 86)
           (CASE-RESPONSE-COPY 38 i i)
           (IF-LESS 57 33 60 i)
           (ADFO i 39 6))
   (CASE-RESPONSE-COPY
    (TRANSPOSE i i i)
    (IF-LESS i 67 94 86)
    95))
(ADF1 78 86 41)
(DO-TIMES (IF-LESS
           20
           (DO-TIMES 10 i)
           (TRANSPOSE i 11 i)
           (CASE-RESPONSE-COPY i 63 i))
          (COPY 28 (ADFO 67 i i) (+ i i))))
```

FIGURE [17]: Example of genetic programming [Koza 1992] in Spector's system [Spector 1995, pp. 5].

The GP system is run with the networks as fitness functions, generating response-producing programs.

They use a three stratus network with 192 inputs units (one for every note value and one for each articulation), 2 outputs units and an only layer of 96 hidden units. They trained the networks with four different categories of inputs:

"[...] The first category consisted of two-measure fragments of Charlie Parker melodies, the second consisted of single measures of Charlie Parker followed by single measures of silence, the third consisted of single measures of Charlie Parker followed by single measures of random melody, and the fourth consisted of single measures of Charlie Parker followed by reversed and randomly manipulated Charlie Parker continuations [...]" [63]. Their networks are trained to reply in a clear way the inputs for the first category. The goal is to train the networks to identify logical continuations to logical parts of jazz melodies.

Also, Biles realizes a version of this system[64] that incorporates a Multi Layer Perceptron with three strata for the evaluation. This system uses a set of high level musical parameters extracted from sequences generated in order to train the ANN. An integer representation of this system is explained in section 3 (*Different chromosome representations*).

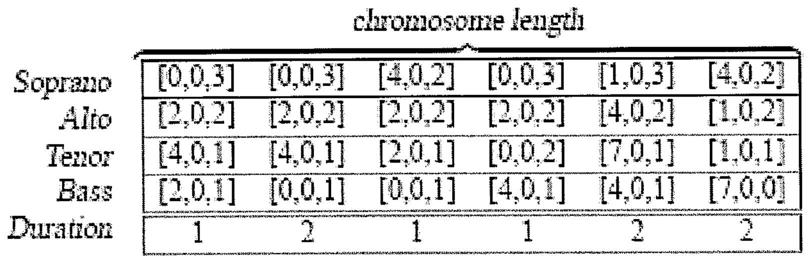
[63][Spector 1995]

[64] [Biles 1994].

2.1.3 Rule-Based Systems

In the rule-based systems the critic is built upon a set of rules that directs the system. This set of rules is obtained by means of musical knowledge or musical research.

Examples about these systems are located in Wiggins and Phon-Amnuaisuk's works[65]. They harmonize chorales and utilize as reference the soprano's melody that the user contributes. The system makes another three voices that would be the alto, tenor and bass. The notes are represented according to the scale and the octaves are differentiated through integer associated. It utilizes operators adapted to the musical domain. The next picture shows a diagram of a Four-Voice Harmony Chromosome.



	chromosome length					
Soprano	[0,0,3]	[0,0,3]	[4,0,2]	[0,0,3]	[1,0,3]	[4,0,2]
Alto	[2,0,2]	[2,0,2]	[2,0,2]	[2,0,2]	[4,0,2]	[1,0,2]
Tenor	[4,0,1]	[4,0,1]	[2,0,1]	[0,0,2]	[7,0,1]	[1,0,1]
Bass	[2,0,1]	[0,0,1]	[0,0,1]	[4,0,1]	[4,0,1]	[7,0,0]
Duration	1	2	1	1	2	2

FIGURE [18]: Schematic Diagram of a Four-Voice Harmony Chromosome. [Phon- Amnuaisuk 1999a, pp. 3]

In their implementation, they use different reproduction operators[66], and here, it is described in musical terms[67]:

<u>Splice</u>: Selects a crossover point amid followed notes of a melody and parallel chords.

<u>Perturb</u>: Mutate by allowing bass, tenor and alto to move up or down by one tone o semitone.

<u>Swap</u>: Mutate by interchanging two randomly selected voices between bass, tenor or alto. This creates the effect of changing the chord between distinct closed and open positions.

<u>Rechord</u>: Mutate to a distinct chord kind. This type of mutation generates a new chord starting in the melody data. This chord is created with the soprano note as base, 3rd or 5th. Doubling can be in any location.

<u>PhraseStart</u>: Mutate the start of every phrase to begin with root position on a down beat.

---

[65] [Wiggins 1999], [Phon- Amnuaisuk 1999a], [Phon- Amnuaisuk 1999b]
[66] Crossover and mutation.
[67] [Phon- Amnuaisuk 1999a]

PhraseEnd: Mutate the ending of every phrase to conclude with a chord in root position.

In the GA configuration, the strings were initialized by randomly selected chords including the soprano pitch. Moreover, a population size over 50 was utilized, with the use of binary tournament selection.

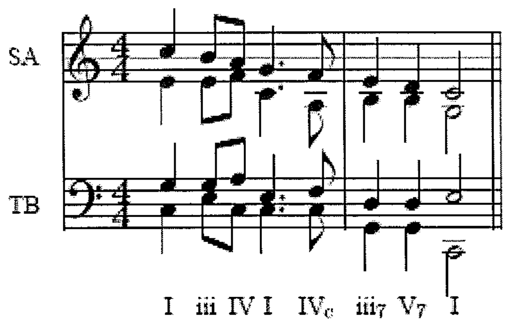


FIGURE [19]: Harmonisation of the first line of *Joy to the World* [Wiggins 1999, pp. 7].

In figure [19] the harmonisation of the initial eight notes of "Joy to the World", carried out by the system, is shown. Different experiments were performed with assorted GA parameter settings, as shown in figure [18]. Other parameters, as mutation rate, crossover rate and other selection schemes, seem to influence the time taken for the population to converge, but do little for the solution quality. This is because the fitness function determines the knowledge in the system concerning what does compose a good song or does not compose a good piece of music; the other parameters determine the search strategy.

A similar system is showed by McIntyre[68]. In this system the musical pieces are harmonized in accordance with the baroque harmony at four voices, using a voice inserted by the user. One power of the algorithm is the ability to find several credible harmonies for a given melody. The evaluation of the function contains three different layers. The first layer analyzes the correction of the chord, the second one examines the harmonic displacement amongst notes, and the last one the softness of chord changes.

[68] [Mc Intyre 1994]

Another important author is Andrew Horner. He dedicates oneself principally to apply the GAs to musical composition. Horner and Goldberg's work[69], constitutes a minimalist musical composition system. This system consists of sequences of simple operations that take an input composition and an output composition provided by the user. The system produces a series of transformations between both compositions by means of an evolutionary procedure. The operations can be note insertion, rotation and deletion. This process follows a series of rules, concretely two, that determine the musical links between the two sequences: the higher the scores obtained by an individual are, the nearer is the result note group to the desired note group, and the closer is the actual number of alteration steps to the wanted duration.

2.1.4 Coevolution and Cellular Automata:

In this case, the musical composition develops in an own way which does not necessarily coincide with the human aesthetics.

In this kind of system, Peter M. Todd and his collaborator's work[70]could be mentioned. This work develops a system based on the co-evolution, existing a set of elements that acts as referee and others that act as compositions creators and both evolve jointly. They research into coevolving individuals to produce rhythmic sequences and other sequences to value them. In the preliminary model, they coevolve simulated 'males' who generate rhythmic songs together with selective 'females' who judge those songs and utilize them to determine whom to pair with. The male song inventors and female song critics utilize neural networks to direct their behaviour.

This neural model has a threshold, but summation of activation coming in the unit takes place over multiple time steps. If the threshold is obtained, the unit shoots for one time step, and its saved input is reset to zero. The output of every neuron is binary. The output signals use one time step to generate a connection between neurons.

In a more recent simulation, they invent 'dumbed-down' male singers, each of whom has genes that in a direct manner encode the notes of the composition. Each male song is

[69] [Horner 1991]
[70] [Todd 1999], [Werner 1997]

composed of 32 notes. These notes can be an only pitch chosen from a two-octave[71] range. Female's genes encode a transition matrix that is used to evaluate transitions between notes in male songs. The matrix is an N-N table, and N is the number of different possible pitches the males can generate.

The population is composed of melody breeding males and females that would couple with the males depending on how the females evaluate male songs. Females evaluate songs in three distinct ways. In the first method, the female evaluates the transition as it takes place in the song: she compares to what extent she expected that transition and she incorporates it to the total score for the song. In the second method, a whole song has to be listened by the female, who counts the different transition classes that appear in the song. As a result, she builds a transition matrix for the song. Finally, she has to compare the transitions in the matrix she has created with those expected ones. Therefore, the highest score will be given to the song with the best. The third method is similar to the first one, but preference is given to "surprising" results, those which did not have a big probability in the female's matrix.

Other works employ techniques related to evolutionary systems like artificial life and cellular automata. Eduardo Miranda[72] shows a granular synthesis system in which the user explains a series of parameters like number of oscillator, waves to use, etc.

Granular synthesis labours by generating a fast series of very brief sound bursts called granules that simultaneously form big sound events. The effects tend to show a huge sense of movement and sound flow.

2.1.5 Hybrid Systems

Hybrid systems use more than one approximation in one system simultaneously (a combination of AI strategies). And normally, these systems combine connectionist and evolutionary methods. Spector and Alpern[73] utilize genetic programming to invent a system with an ANN as a fitness evaluator of the response to a measure "call". Biles, in 1996, in a new attempt to improve the efficiency of his old system[74], uses an ANN as

[71] 24 pitches
[72][Miranda 1995]
[73] [Spector 1995b]
[74] [Biles 1994]

fitness function too, but it has no excessively good results. Burton and Vladimirova[75] use an ART[76] ANN to allot fitness measures to different rhythms produced by a GA. This ART networks are efficient pattern recognizers. They classify patterns and identify pattern categories without necessity of supervision. The fitness is allotted based on resemblance to patterns, with new patterns stipulated for individuals that are in a sufficient manner different from existing patterns.

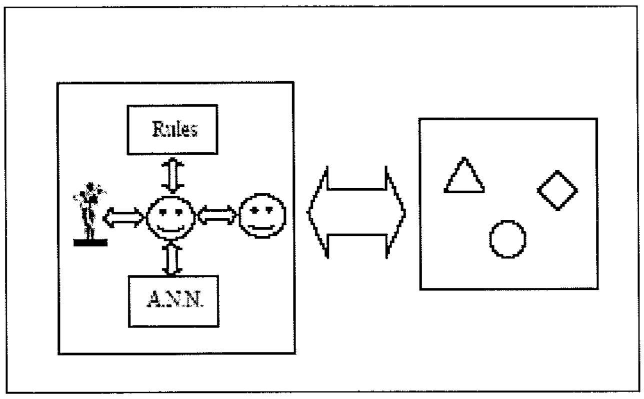


FIGURE [20]: [Santos 2000, pp. 4]
LEFT: Different models in a single system.
RIGHT: Integration of methods from distinct approaches in a common field

3. Different Chromosome Representations

In the following picture there is an integer representation of Biles' system[77]. The number on the left of the thick line is the fitness value, and the numbers on the right represents the chromosome. The phrase 23 has a fitness value of -12. Its chromosome is the series of four numbers, and each number is a pointer to the population. The population sizes are 64 measures and 48 phrases, and the chromosome of four pointers need 24 bits.

[75] [Burton 1997]
[76] Adaptive Resonance Theory
[77] [Biles 1994].

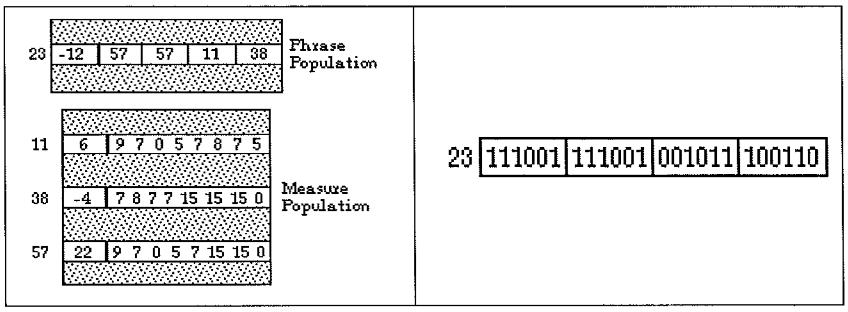


FIGURE [21]: Phrase and its Measures. Chromosome at the Bit Level, [Biles 1994, pp. 5]

Ariza[78] presents us the information as a list of three integers: divisor, multiplier and note/rest-state. The duration of the rhythm-tuples in a chromosome is quantified in reference to a beat-time or the valuation of pulse calculated in seconds. For the calculation of the duration of the rhythm-tuple there are two steps:

First, divide the beat-duration by the divisor, afterwards multiply it by the multiplier. The note/rest-state settles if the measured duration is a rest (0) or a note (1).

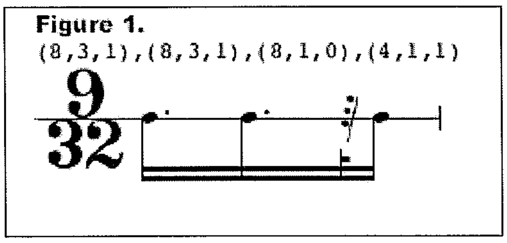


FIGURE [22]: Chromosome illustrated as a list of rhythmic-tuples and as a notated rhythm. [Ariza 2002, pp. 2]

Papadopoulos[79] uses a degree-based representation. The chromosome represents the degrees of the scale, concerning the prevalent chord. This approach uses the union of the degree and its analogous chord to indicate the actual pitch of the melody. Besides, he uses an extended-degree representation that consists of 21 distinct values which fit to 3 octaves (for a 7 note scale). Therefore, the chromosome is a series of extended-degree duration pairs, rest been discerned by the constant rest instead of the extended-degree.

78 [Ariza 2002]
79 [Papadopoulos 1998].

In Gartland Jones' system[80], the genotype is a bit array that includes meta-data attributes and a simple model is described next. It is constituted by two parts.

The section object is the external object typifying a piece of music. It has an array of notes and characteristics for target match fitness.

The note object gives the values for a single note. It has target fitness (0-1) and characteristics for pitch, duration and velocity.

In Manzolli's approach[81] the individuals are defined as chords of four notes. The notes of these chords are randomly produced on the interval [0,127] (corresponding with the MIDI note number). In every generation, a population of 30 chords is created and valued. Internally, the chords are represented like a chromosome with 28 bits, compound of 4 words with 7 bits.

1011111	1010111	0010111	0100111

FIGURE [23]: The structure of a MIDI chromosome. [Manzolli 1999, pp.2]

In Pigg's work[82], an individual includes two groups of genomes. The first is associated to the notes in the measurements, and it is composed of 12 possible notes, a rest and a hold. The hold is used instead of a note, to permit distinct note-lengths in the song.

The second is associated to the octaves of the notes, and its alphabet is composed of four octaves and a null octave, to fit the spaces in the measures where a rest or an extend is exhibited.

In Jacob's implementation[83], the alleles represent vertical pitch combinations. They are similar to interval types, but they add any number of pitches (from 1 to 12). Every allele is twelve bit long, symbolizing a group of semitones that can be played at the same time. Each pair of contiguous alleles shows a valid transition. Like interval types, the twelve transpositions of a valid pitch or the transition between two combinations, are allowed.

The system of Wiggins[84]et al. uses for the implementation three sections of the GA:

[80] [Gartland-Jones 2003a]
[81] [Manzolli 1999]
[82] [Pigg 2002]
[83] [Jacob 1994]
[84] [Wiggins 1999]

Chromosome representations: Chords and keys are the principal concepts in harmonisation of occidental tonal music. To express the twelve semitones, the standard five accidentals are used. To distinguish the different octaves an associated integer is used, and the time intervals are represented as integers too. The chromosome representation is made by means of a matrix.

Reproduction operators: In this implementation different kinds of mutation and crossover operators are used (splice, perturb, swap, rechord, and phrase start, phrase end[85]).

Fitness function: Wiggins et al. do not permit progression to dissonant chord and leap of minor and major 7ths of diminished and augmented intervals and of intervals bigger than one octave. Among voices, they avoid parallel perfect 5ths, parallel unison, and parallel octaves. They prohibit progression from diminished 5th to perfect 5th and crossing voices.

4. Summary of Former Results

Genetic algorithms have been utilized to produce music structures based on different approaches.

Horner and Goldberg[86] use genetic algorithms for thematic bridging among simple melodies, and their work is revised by Todd and Werner[87]. McIntyre[88] produces a four part Baroque harmonisation of an input melody. Both use a knowledge-based fitness function.

Other authors use a human user as a fitness rater: Jacob[89] designs a composing system using material produced from phrases supplied by the user as building blocks. Horowitz[90] employs interactive genetic algorithms to produce rhythmic patterns. Ralley[91] proposes a composition system that utilizes a strategy of data diminution to comprise two extremes of individual fitness task and to elaborate melodies. Biles[92]develops a system which uses GA

85 These operators have been explained in the section 2.1.3 (pp.27).
86 [Horner 1991]
87 [Todd 1999]
88 [Mc Intyre 1994]
89 [Jacob 1994], [Jacob 1996]
90 [Horowitz 1994]
91 [Ralley 1995]
92 [Biles 1994]

to imitate a jazz musician learning to improvise. Moroni[93] evolves a system established on evolutionary computation strategies for composing music in real time.
In other kind of system that uses a neural network as fitness function, Gibson and Byrne's works can be mentioned[94], who make simple harmonizations, using the tonic subdominant and dominant chords. Spector and Alpern[95] use GP[96] to produce a one measure reply to a one measure "call", by means of a neural network as a fitness estimator of the reply. Biles[97] tries to use a neural network to improve his first work. Papadopoulos[98] designs a system for the creation of jazz melodies over an input chord sequence. Burton and Vladimirova[99] propose a method based on the clustering behaviour of an ART (Adaptive Resonance Theory) to solve the problems of neural fitness operators.

Johanson and Poli[100] produce melodies by means of genetic programming. They use an interactive system which permits users to develop little musical sequences and they present an extension which utilizes a neural network to model the user's choice.
Johnson[101] develops a group of parameters for a granular synthesis machine.

5. Comparison of Genetic Operations

5.1 Fitness Function Selection

The algorithm fitness function is a set of melodies. It is important to bear in mind that the aim of this work is not to produce melodies. The actual goal is to compare the different kinds of genetic operations (short GO). These GOs have been divided in three groups, defined in the following lines.

- **Class 1:** Classic genetic operations.

[93] [Moroni 2000]
[94] [Gibson 1991]
[95] [Spector 1995b]
[96] [Koza 1992])
[97] [Biles 1996]
[98] [Papadopoulos 1998]
[99] [Burton 1997]
[100] [Johanson 1998]
[101] [Johnson 1999]

- Mutation.
- One point crossover + mutation.
- Two point crossover + mutation.

- **Class 2:** Classic genetic operations mixed with musical functions.
  - One point crossover with:
    - transpose and classic mutation.
    - transpose and rhythm mutation.
    - reverse and classic mutation.
    - reverse and rhythm mutation.
    - mirror and classic mutation.
    - mirror and rhythm mutation.
  - Two point crossover with:
    - transpose and rhythm mutation.
    - reverse and classic mutation.
    - reverse and rhythm mutation.
    - mirror and classic mutation .
    - mirror and rhythm mutation.
  - One point crossover with mirror, reverse + mutation.
  - Two point crossover with mirror, reverse + mutation.
  - One point crossover with mirror, reverse, transposition + mutation.
  - Two point crossover with mirror, reverse, transposition + mutation.
  - One point crossover with mirror, reverse + rhythm mutation
  - Two point crossover with mirror, reverse + rhythm mutation.
  - One point crossover with mirror, reverse, transposition + rhythm mutation.
  - Two point crossover with mirror, reverse, transposition + rhythm mutation.

- **Class 3:** Musical functions
  - Different kinds of mirror functions + mutation / rhythm mutation.

- Different types of reverse functions + mutation / rhythm mutation.
- Mirror and reverse + mutation / rhythm mutation.
- Different kinds of swap between notes + mutation / rhythm mutation.
- Transpose + mutation / rhythm mutation.

The fitness function consists of seven already composed melodies with similar structure and tonality, which is taken as a reference to give a fitness value for each melody of the population. The fitness value measures the similarity of the output of the GA and these melodies. By means of this fitness value the different operation classes can be compared.

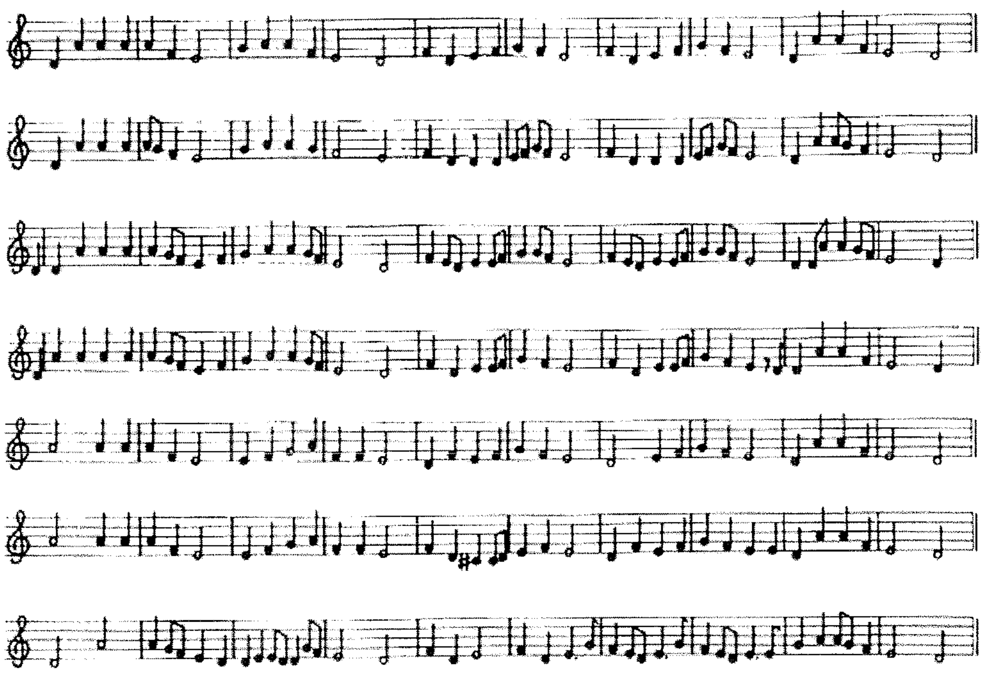


FIGURE [24]: Fitness function melodies. [Sundberg 1993, pp. 278]

Song variations are especially suitable to be used as fitness function because they have:

1. Same length.
2. Same tonality.
3. Similar structure.

Due to this reason, the fitness function is made up of seven variations of the Swedish folk song "Ro, Ro till Fiskeskär" used in a paper from Sundberg[102] concerning the application of generative grammars for producing Swedish nursery tunes.

All the variations have the same number of bars, which is an important aspect because, in this way, this feature allows to represent all the songs in arrays with the same size and the same number of integers.

As it was previously mentioned, all of them are similar, because there are variations of a unique song. This is interesting, taking into account that the same structure is needed.

In these songs, a preference for half notes at the end of every melody is noticeable. The entire songs finish on the fundamental of the tonic and all the even-numbered bars conclude on this note of the fifth of the dominant.

5.2 Chromosome Representation[103] and Main Characteristics

- Number of fitness function melodies: 7.
- Number of bars: 10 (4/4).
- Number of integers per bar: 8 integers
- Minimum duration of a note: quaver. This will be the smallest unit.
- Maximum duration of a note: minim.
- Lowest note: C#4 (61 midi-key-number)
- Highest note: A4 (69 midi-key-number)
- Integer representation of the notes:

Integer	Pitch	Midi-key-number
0	C#4	61
2	D4	62
4	E4	64
6	F4	65
8	G4	67
10	A4	69
12	Rest	-
14	Prolongation	-

FIGURE [25]

[102] [Sundberg 1976]

[103] Even numbers where selected to allow easier later extensions of the program, for example to enlarge the range pitch. If all the integers between 0 and 10 were selected, it would be exactly one scale. Then, the rest and the prolongation values would continue being the same values that in this implementation.

- Examples:

  0,2,4,6 = C#4, D4, E4, F4 (all quavers).

  0,14,4,6 = C#4 (quarter note), E4 and F4 (quavers).

  0,14,14,14 = C#4 (minim).

  0,12,10,14 = C#4 (quaver), rest (quaver), A4(quarter note).

- Each melody will have 80 integers. (8 integers x 10 bars)
- Example of a fitness function melody (number 1, figure 24):

  2,14,10,14,10,14,10,14 – 10,14,6,14,4,14,14,14 – 8,14,10,14,10,14,6,14 – 4,14,14,14,2,14,14,14 – 6,14,2,14,4,14,6,14 – 8,14,6,14,4,14,14,14 – 6,14,2,14,4,14,6,14 – 8,14,6,14,4,14,14,14 -- 2,14,10,14,10,14,6,14 – 4,14,14,14,2,14,14,14.

5.3 The Fitness Value

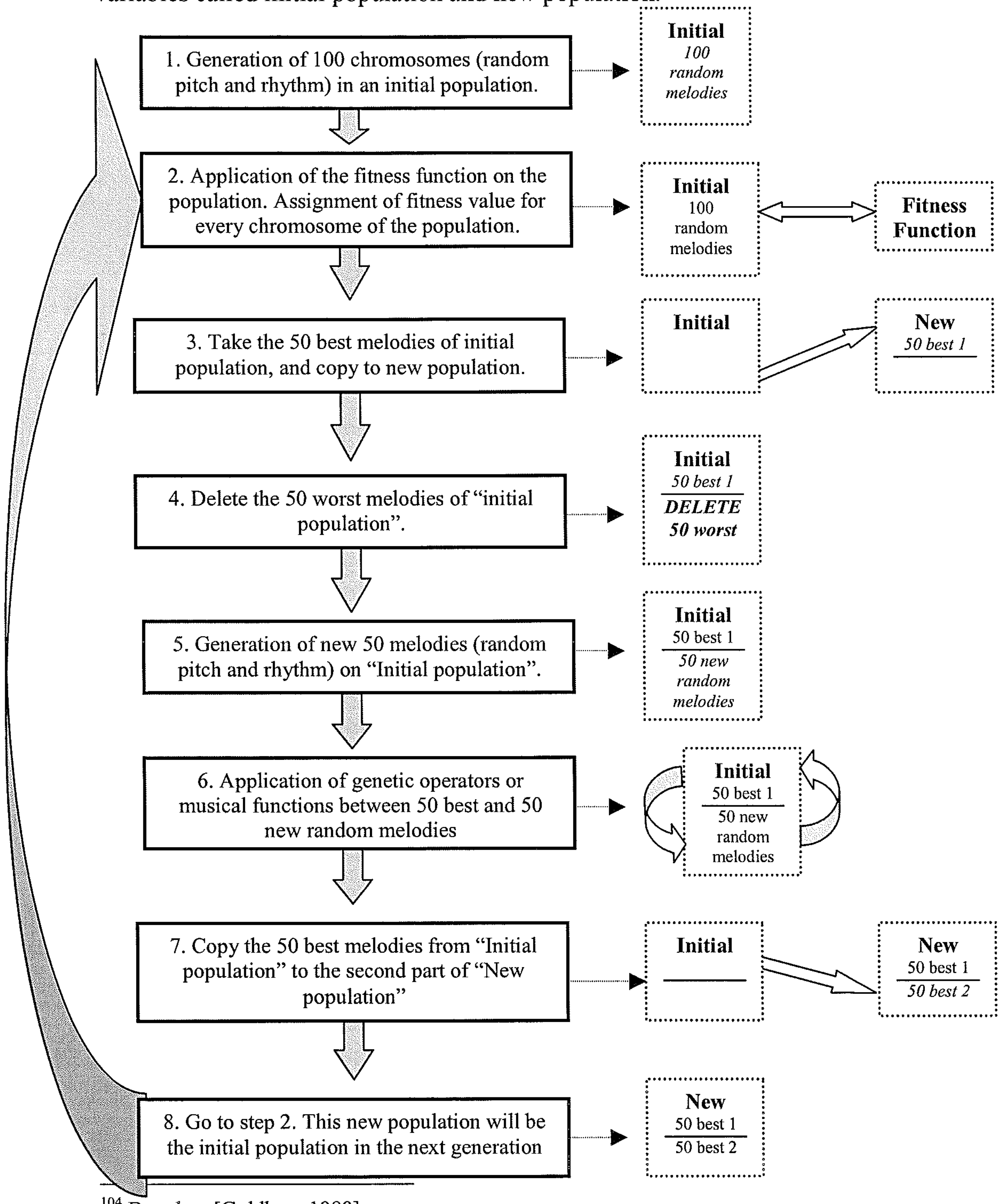
The program uses a function to assign a fitness value to every melody. This function does a note to note comparison with the seven melodies of the fitness function, which consists of 80 integers each one. Therefore, if a melody gets a fitness value of 80, this means that a melody has been found.

The function takes the first melody, and compares it with every melody of the fitness function. Hence, it will have 7 fitness values. This process is repeated with all the melodies of the population. At this stage, there are 7 fitness values for every melody. The best melody will be the one which has the highest fitness value.

5.4 Application of the Genetic Algorithm

5.4.1. Design of the Algorithm[104]

Example with a population size of 100 chromosomes (i.e. melodies), starting from two variables called initial population and new population.



1. Generation of 100 chromosomes (random pitch and rhythm) in an initial population.

2. Application of the fitness function on the population. Assignment of fitness value for every chromosome of the population.

3. Take the 50 best melodies of initial population, and copy to new population.

4. Delete the 50 worst melodies of "initial population".

5. Generation of new 50 melodies (random pitch and rhythm) on "Initial population".

6. Application of genetic operators or musical functions between 50 best and 50 new random melodies

7. Copy the 50 best melodies from "Initial population" to the second part of "New population"

8. Go to step 2. This new population will be the initial population in the next generation

[104] Based on [Goldberg 1989]

5.4.2 How are the genetic operations applied?

If crossover (one and two point) is applied it works the following way. If the population size is 100 (where the chromosomes from 1 to 50 are the best, and from 51 to 100 are random), the crossover is applied between the chromosome nº1 and nº51; Chr2 and Chr52; Chr3 and Chr53...

If the genetic operations are used after one point crossover, they will be applied from the randomly chosen point until the end of the chromosome. In the two point crossover the application affects the selected area (see below).

<u>Class 1. Example of a two point crossover</u>

The program selects two random positions, for example Pos1 = 4, and Pos2 = 7. Then:

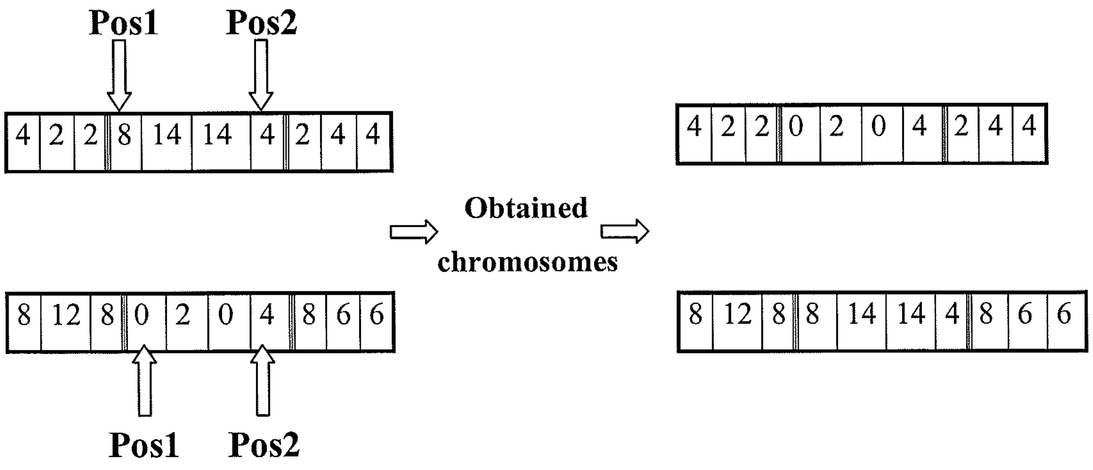


FIGURE [26]

<u>Class 2. Example with two point crossover and the reverse, transpose and mirror functions[105]</u>

The two point crossover operations is applied in the same manner that in the first example, therefore, starting from the chromosomes obtained. Afterwards the reverse is applied only in the part situated between the two randomly selected positions.

[105] The functions are every time applied in the order as they appear in the command line

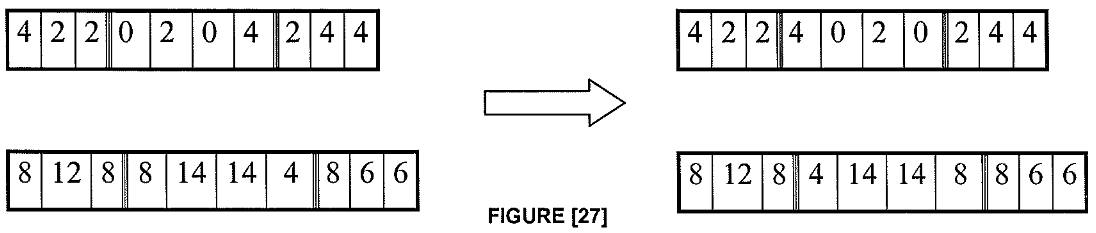


FIGURE [27]

The next step is the transposition of one value randomly selected by the program between 0 and 6 (range pitch).If the transpose value applied to a chromosome is 6, the resultant chromosome would be equal to the initial chromosome. The following example[106] shows the results for a transpose value = 1:

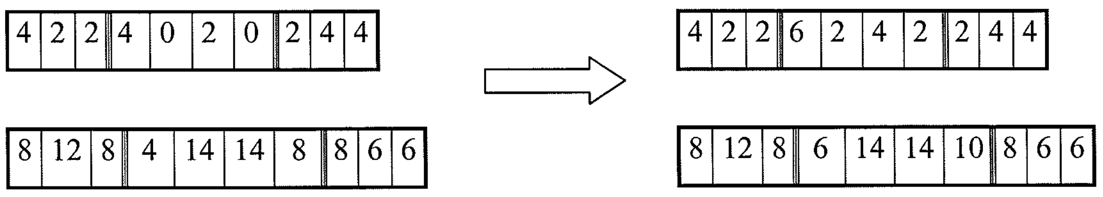


FIGURE [28]

The last step is the mirror function application. The mirror axis will be the first note between Pos1and Pos2. As its name denotes, the axis functions as a "mirror": the notes under the axis are placed over it, and vice versa, always keeping a constant distance between the notes and the axis.

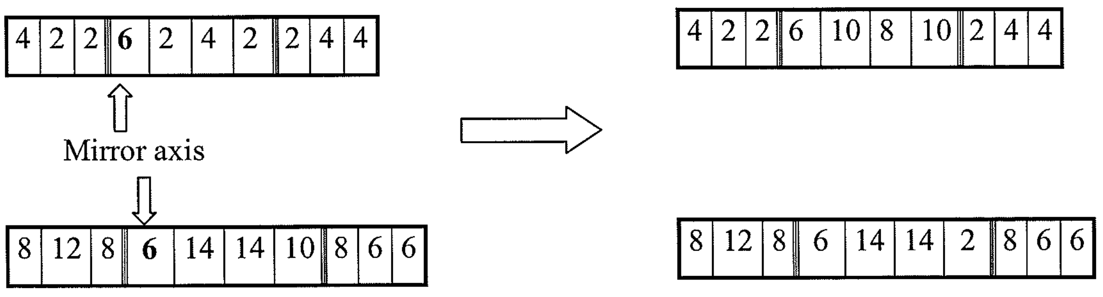


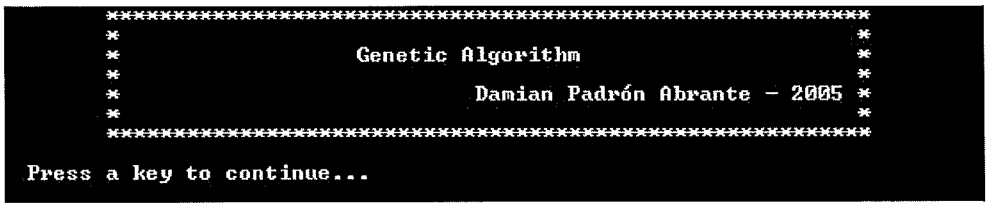
FIGURE [29]

[106] See representation of the notes values, figure 25, pp. 36.

<u>Class 3.</u>

The study of the behaviour is not the main goal of this work, but it is important to know that this musical functions are applied to the whole population[107] without any crossover operation. The explanation of every kind of function is given in the program menu.

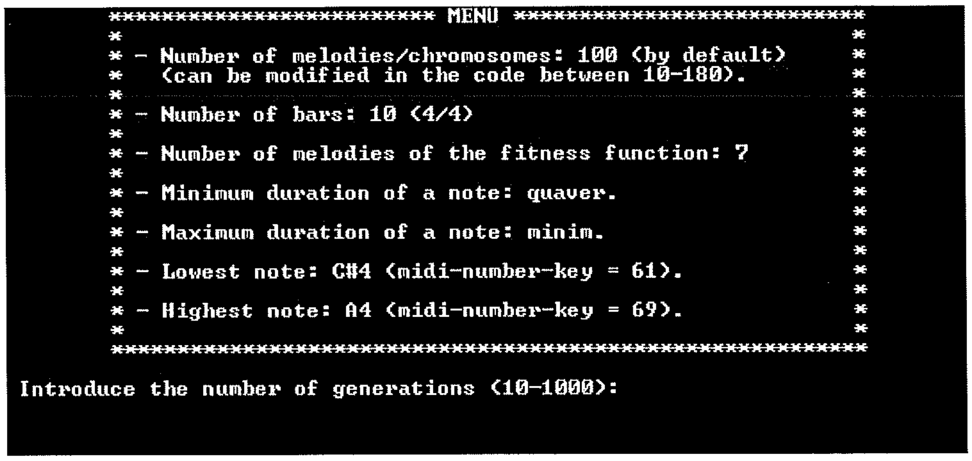
5.5 How does the program work?



```
***********************************************************
*                                                         *
*                   Genetic Algorithm                     *
*                                                         *
*                         Damian Padrón Abrante - 2005 *
*                                                         *
***********************************************************

Press a key to continue...
```

FIGURE [30]

1. After this presentation screen, when the program is executed, the first menu (see figure 31) displays some of the main algorithm features, and the user has the possibility of choosing the number of generations. This number is recommended to be between 10 and 1000, but the user can test it with bigger numbers.

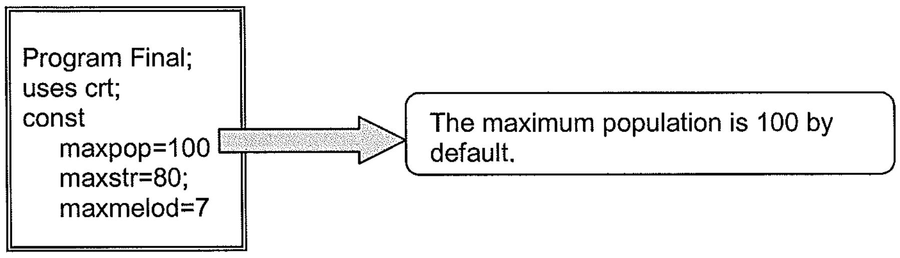


```
************************* MENU ****************************
*                                                          *
* - Number of melodies/chromosomes: 100 (by default)       *
*   (can be modified in the code between 10-180).          *
*                                                          *
* - Number of bars: 10 (4/4)                               *
*                                                          *
* - Number of melodies of the fitness function: 7          *
*                                                          *
* - Minimum duration of a note: quaver.                    *
*                                                          *
* - Maximum duration of a note: minim.                     *
*                                                          *
* - Lowest note: C#4 (midi-number-key = 61).               *
*                                                          *
* - Highest note: A4 (midi-number-key = 69).               *
*                                                          *
************************************************************

Introduce the number of generations (10-1000):
```

FIGURE [31]

NOTE: Moreover, the population size can be changed, but only in the program Code (line 6, pp. 57). Its maximum value is 180.

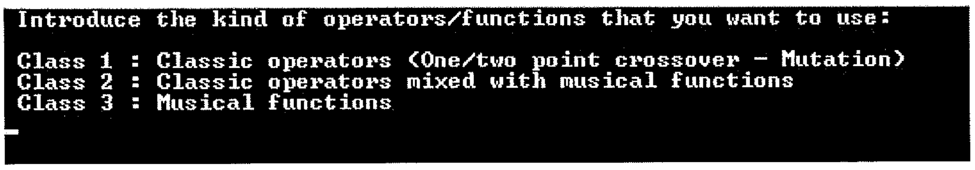
[107] After the application of FF and selection like in the other classes



```
Program Final;
uses crt;
const
    maxpop=100
    maxstr=80;
    maxmelod=7
```

FIGURE [32]

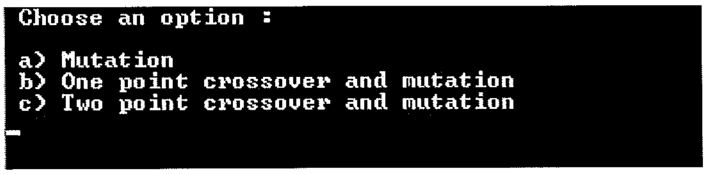
2. The second menu displays the different classes of operations that the user can select:



```
Introduce the kind of operators/functions that you want to use:

Class 1 : Classic operators (One/two point crossover - Mutation)
Class 2 : Classic operators mixed with musical functions
Class 3 : Musical functions
_
```

FIGURE [33]

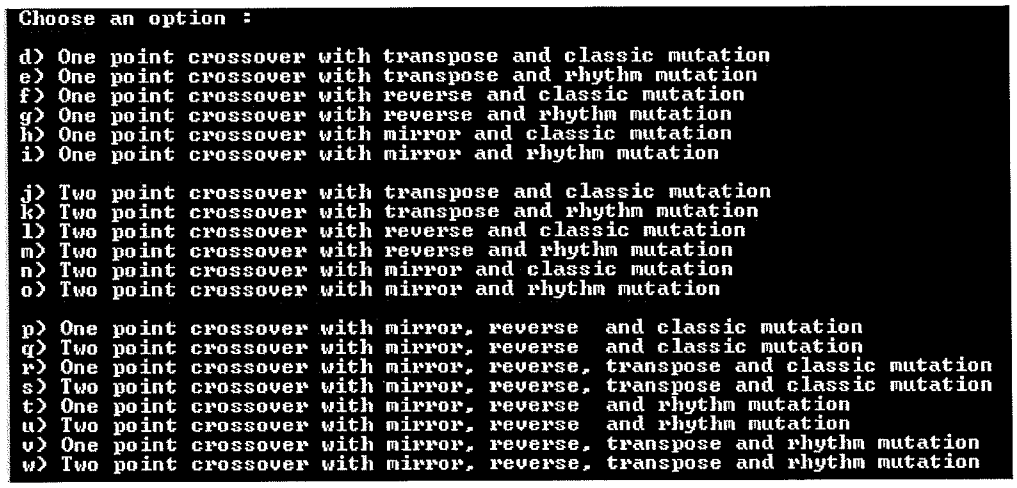
3. a) Class 1: Classic operations. One point crossover and two point crossover are applied between the first 50 % and the second 50 % of melodies.



```
Choose an option :

a> Mutation
b> One point crossover and mutation
c> Two point crossover and mutation
_
```

FIGURE [34]

3. b) Class 2: Classic operations mixed with musical functions.

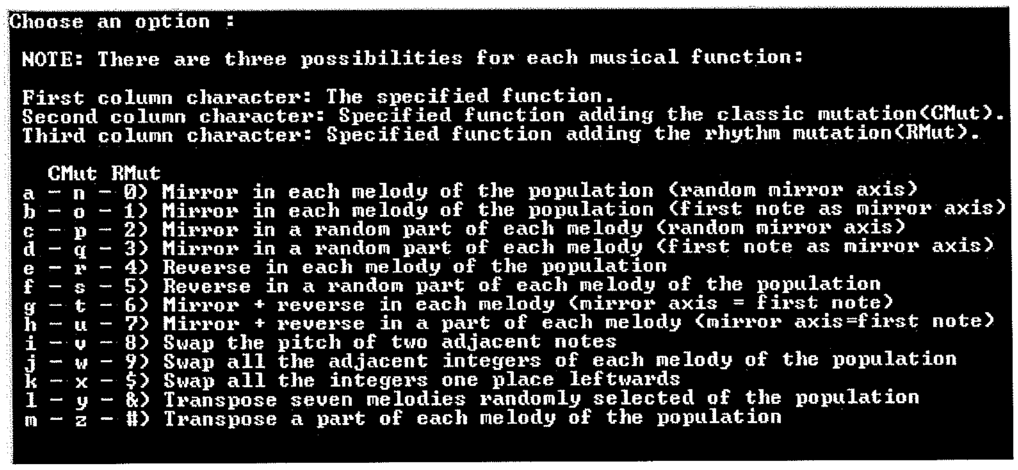


```
Choose an option :

d> One point crossover with transpose and classic mutation
e> One point crossover with transpose and rhythm mutation
f> One point crossover with reverse and classic mutation
g> One point crossover with reverse and rhythm mutation
h> One point crossover with mirror and classic mutation
i> One point crossover with mirror and rhythm mutation

j> Two point crossover with transpose and classic mutation
k> Two point crossover with transpose and rhythm mutation
l> Two point crossover with reverse and classic mutation
m> Two point crossover with reverse and rhythm mutation
n> Two point crossover with mirror and classic mutation
o> Two point crossover with mirror and rhythm mutation

p> One point crossover with mirror, reverse  and classic mutation
q> Two point crossover with mirror, reverse  and classic mutation
r> One point crossover with mirror, reverse, transpose and classic mutation
s> Two point crossover with mirror, reverse, transpose and classic mutation
t> One point crossover with mirror, reverse  and rhythm mutation
u> Two point crossover with mirror, reverse  and rhythm mutation
v> One point crossover with mirror, reverse, transpose and rhythm mutation
w> Two point crossover with mirror, reverse, transpose and rhythm mutation
```

FIGURE [35]

3. c) Class 3: Musical functions. Within this class, there are three different possibilities, according to the chosen column character:

- The first one consists of the musical function itself.
- In the second one, the classic mutation is added to the musical function.
- The third one, the rhythm mutation is added to the musical function.



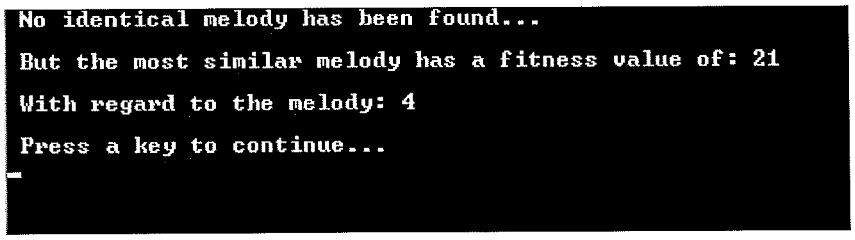
```
Choose an option :

NOTE: There are three possibilities for each musical function:

First column character: The specified function.
Second column character: Specified function adding the classic mutation(CMut).
Third column character: Specified function adding the rhythm mutation(RMut).

  CMut RMut
a - n - 0) Mirror in each melody of the population (random mirror axis)
b - o - 1) Mirror in each melody of the population (first note as mirror axis)
c - p - 2) Mirror in a random part of each melody (random mirror axis)
d - q - 3) Mirror in a random part of each melody (first note as mirror axis)
e - r - 4) Reverse in each melody of the population
f - s - 5) Reverse in a random part of each melody of the population
g - t - 6) Mirror + reverse in each melody (mirror axis = first note)
h - u - 7) Mirror + reverse in a part of each melody (mirror axis=first note)
i - v - 8) Swap the pitch of two adjacent notes
j - w - 9) Swap all the adjacent integers of each melody of the population
k - x - $) Swap all the integers one place leftwards
l - y - &) Transpose seven melodies randomly selected of the population
m - z - #) Transpose a part of each melody of the population
```

FIGURE [36]

4. When the processing time finishes, there are two possible results:

a) No identical melody to the fitness function is found.

In this case the highest fitness value of one melody of the population is given, based on its resemblance to one of the fitness function melodies (the number of this fitness function is also provided).



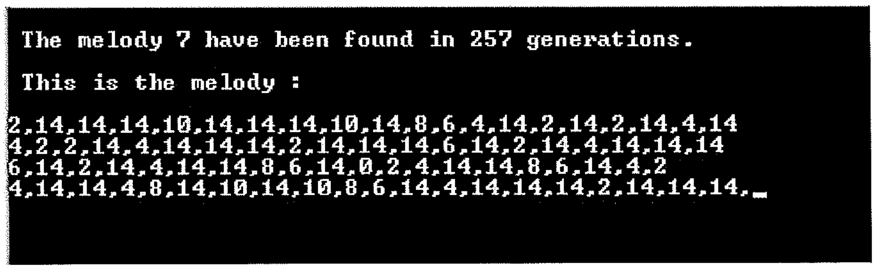
```
No identical melody has been found...

But the most similar melody has a fitness value of: 21

With regard to the melody: 4

Press a key to continue...
```

FIGURE [37]

b) An identical melody to the fitness function is found.

In this case, the melody number and the found melody itself are shown.

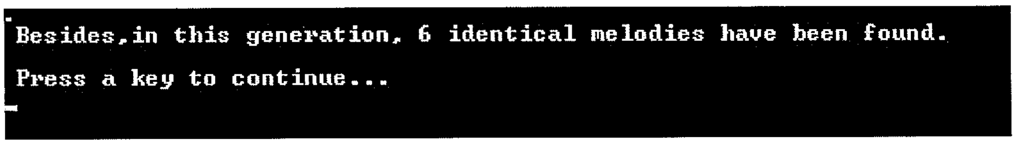
Besides, the number of generations that the program has needed to find a solution is also provided.



```
 The melody 7 have been found in 257 generations.

 This is the melody :

2,14,14,14,10,14,14,14,10,14,8,6,4,14,2,14,2,14,4,14
4,2,2,14,4,14,14,14,2,14,14,14,6,14,2,14,4,14,14,14
6,14,2,14,4,14,14,8,6,14,0,2,4,14,14,8,6,14,4,2
4,14,14,4,8,14,10,14,10,8,6,14,4,14,14,14,2,14,14,14,_
```

FIGURE [38]

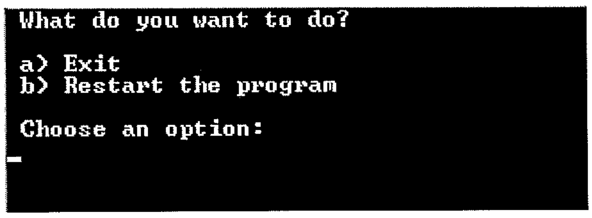
Next, the following screen displays how many identical melodies have been found in the generation where the algorithm found the first solution.



```
Besides,in this generation, 6 identical melodies have been found.

Press a key to continue...
_
```

FIGURE [39]

5. The last screen contains two options:



```
What do you want to do?

a> Exit
b> Restart the program

Choose an option:
_
```

FIGURE [40]

6. Statistical Results

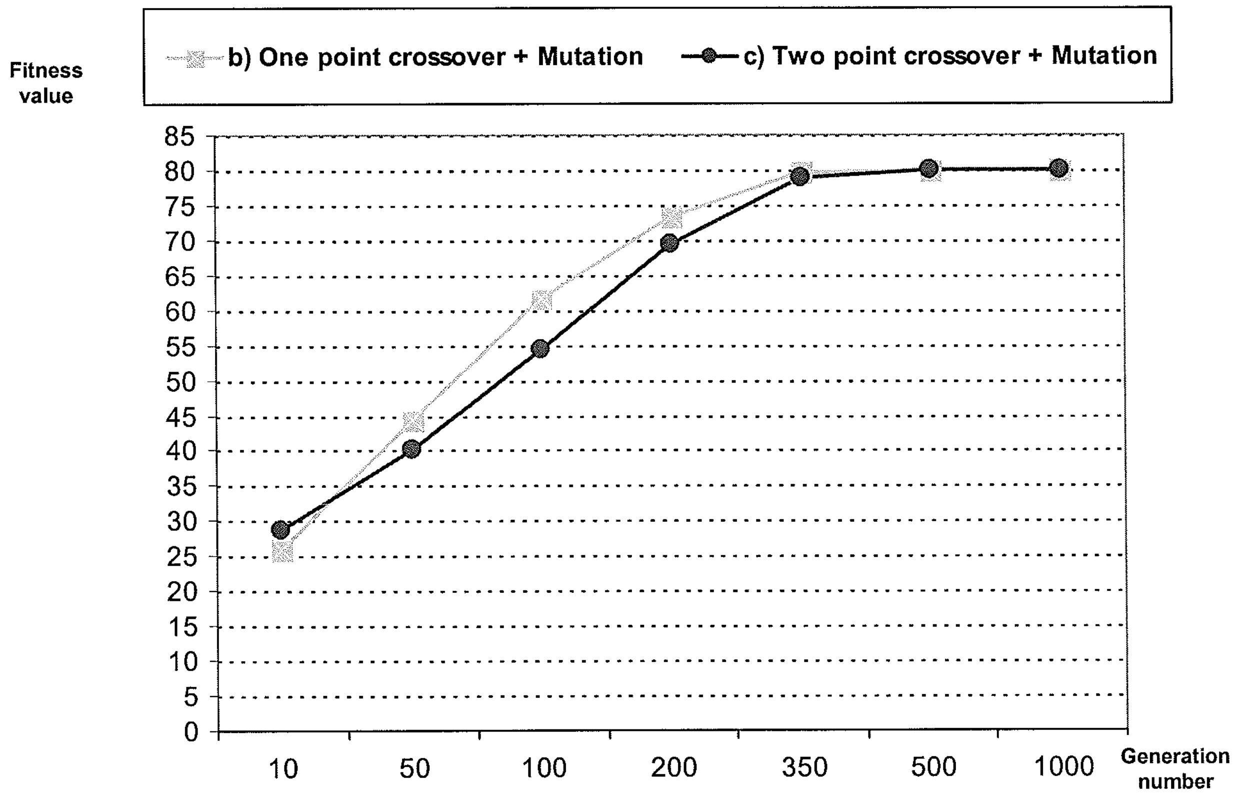
The data showed in this section are the results obtained after testing the algorithm for class 1 and class 2. The selected population size is 100 chromosomes. The program has been run ten times for all the following examples. The values shown in the graphics are the average of the mentioned ten executions. When mutation (only) operation is mentioned, it always performs by altering one randomly chosen integer (like in the classical GA, see figure 13, pp. 16).

The rhythm mutation is performed by two different functions. The first of them, takes groups of three adjacent integers in every chromosome and if none of them has a value=14 (prolongation), it changes the value of the first integer introducing a 14. Otherwise, it changes the first integer giving it a random value. The second function takes a random number of melodies of the population. Afterwards, it selects a random bar and turns the last note into a minim.

For each class of genetic operations / musical function the following information is presented:

- A diagram which shows the results progression as generation number increases.
- Numerical values table.
- A graphic which represents the generation number needed to find a solution.

- Class 1) Classic Genetic Operations



GENETIC OPERATION: \ GENERATION NUMBER	10	50	100	200	350	500	1000
b) One point crossover + Mutation	26,0	44,2	61,6	73,2	79,8	80,0	80,0
c) Two point crossover + Mutation	28,8	40,0	54,4	69,4	79,0	80,0	80,0

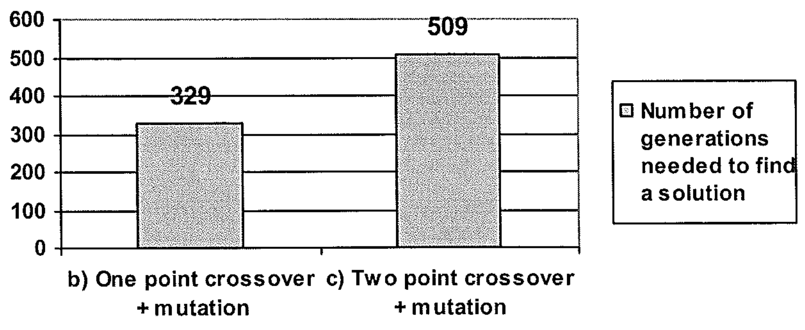
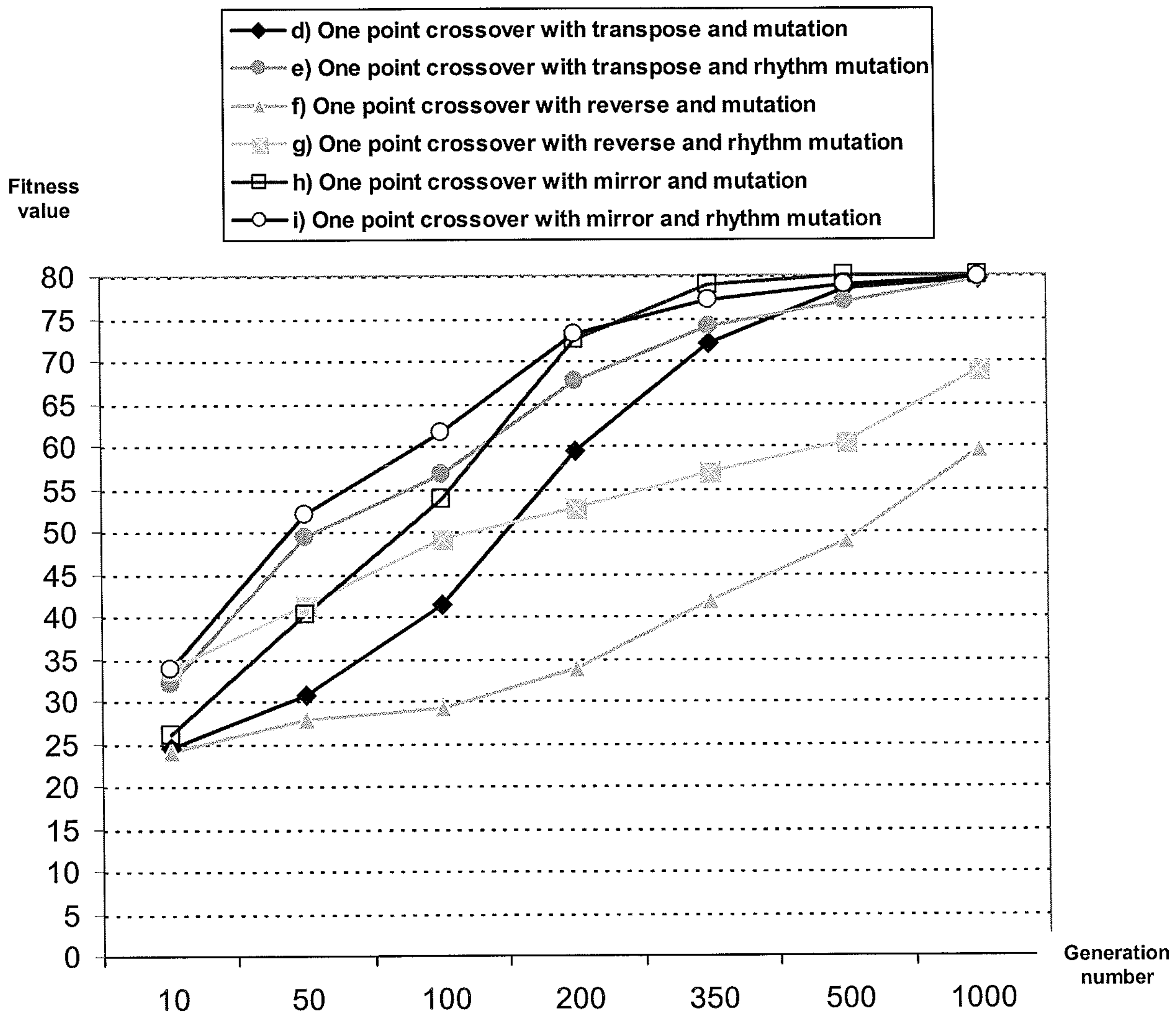


FIGURE [41]

- **Class 2) Musical functions with one-point crossover**



GENETIC OPERATION: \ GENERATION NUMBER	10	50	100	200	350	500	1000
d) One point crossover with transpose and mutation	24,4	30,6	41,4	59,4	72,0	78,4	79,6
e) One point crossover with transpose and rhythm mutation	32,0	49,4	56,6	67,6	74,0	77,0	79,6
f) One point crossover with reverse and mutation	24,0	27,8	29,2	33,8	41,8	48,8	59,6
g) One point crossover with reverse and rhythm mutation	33,8	41,4	49,2	52,6	57,0	60,4	69
h) One point crossover with mirror and mutation	26,0	40,2	53,8	72,4	79,0	80,0	80,0
i) One point crossover with mirror and rhythm mutation	33,8	52,0	61,6	73,2	77,2	78,8	79,8

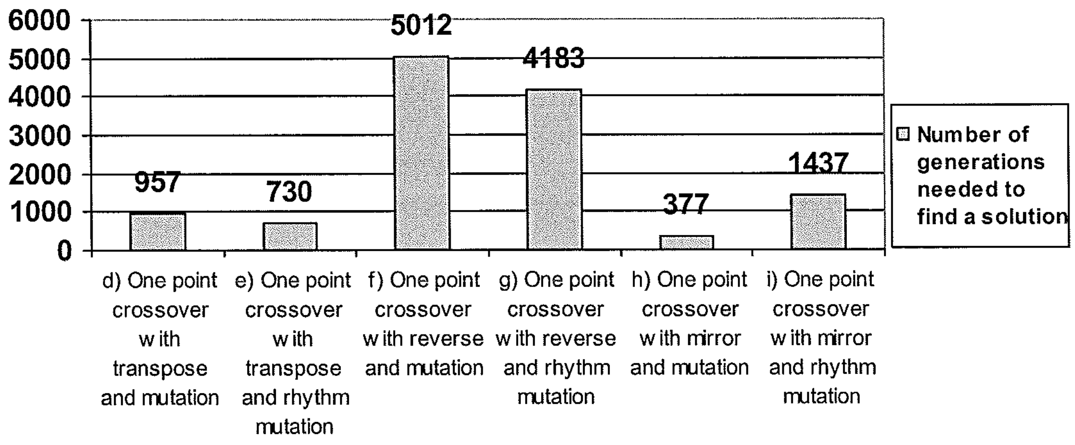
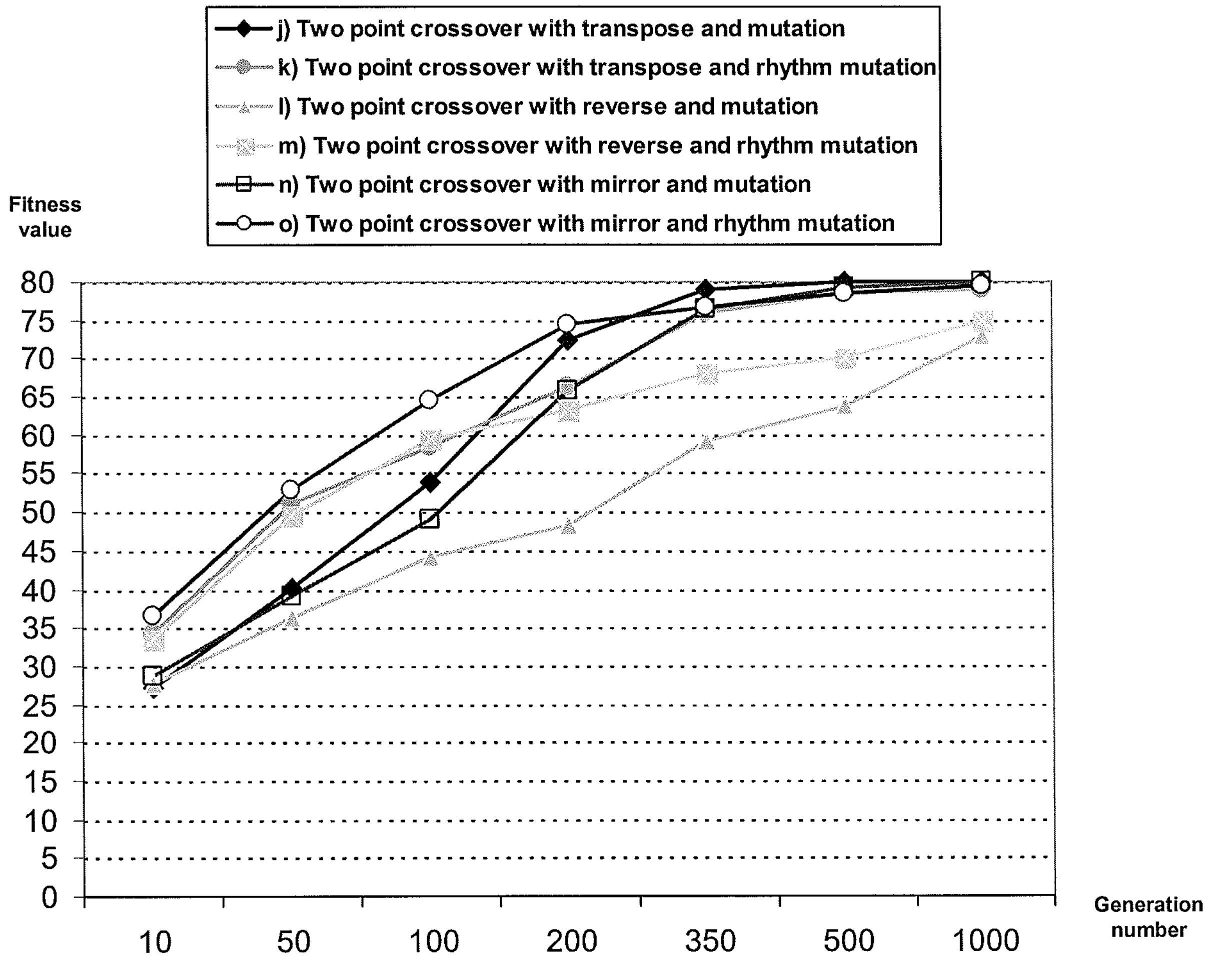


FIGURE [42]

- **Class 2) Musical functions with two-point crossover**



GENETIC OPERATION: \ GENERATION NUMBER	10	50	100	200	350	500	1000
j) Two point crossover with transpose and mutation	27,2	40,2	53,8	72,4	79,0	80,0	80,0
k) Two point crossover with transpose and rhythm mutation	34,2	51,0	58,4	66,2	76,0	78,4	79,0
l) Two point crossover with reverse and mutation	27,6	36,4	44,2	48,2	59,2	63,8	73,0
m) Two point crossover with reverse and rhythm mutation	33,6	49,4	59,4	63,2	68,0	70,0	75,0
n) Two point crossover with mirror and mutation	28,6	39,0	49,0	65,8	76,4	79,2	80,0
o) Two point crossover with mirror and rhythm mutation	36,6	52,8	64,6	74,4	76,6	78,6	79,4

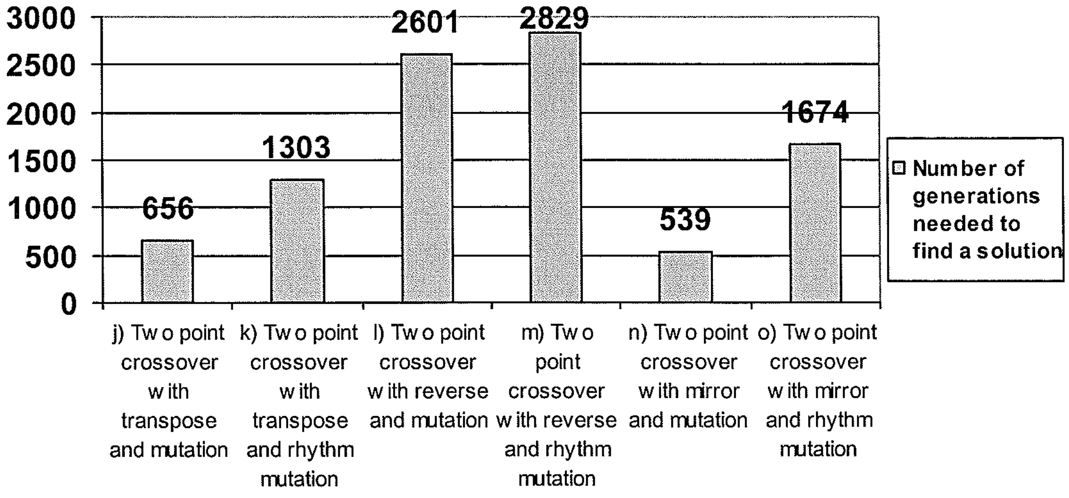
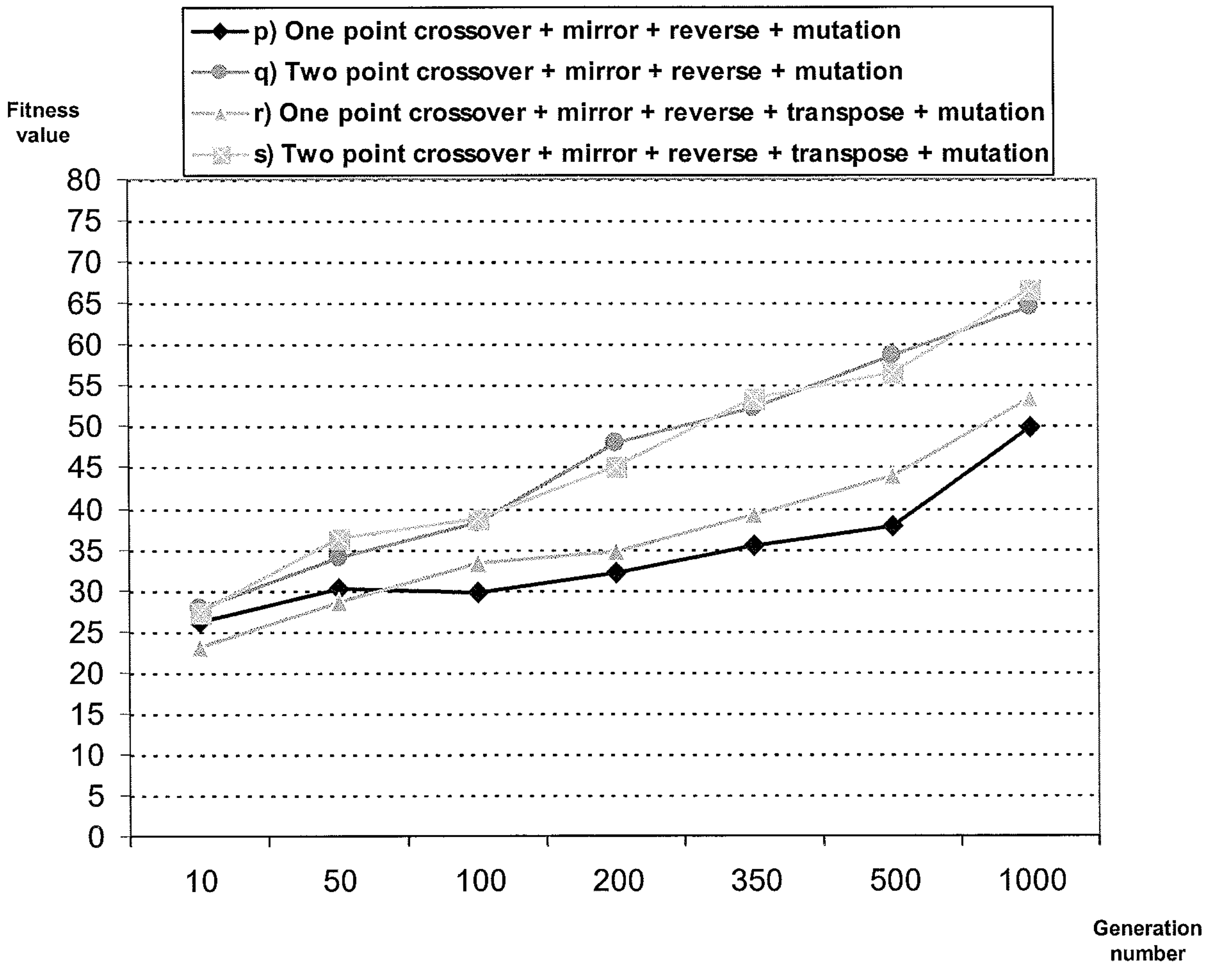


FIGURE [43]

- Serial Application of Class 2 musical functions with One/two point crossover and mutation



GENERATION NUMBER / GENETIC OPERATION:	10	50	100	200	350	500	1000
p) One point crossover + mirror + reverse + mutation	26,2	30,2	29,6	32,0	35,4	37,8	49,8
q) Two point crossover + mirror + reverse + mutation	27,8	34,0	38,2	48,0	52,2	58,6	64,6
r) One point crossover + mirror + reverse + transpose + mutation	23,0	28,4	33,2	34,6	39,2	43,8	53,4
s) Two point crossover + mirror + reverse + transpose + mutation	27,4	36,4	38,6	45,2	53,4	56,6	66,8

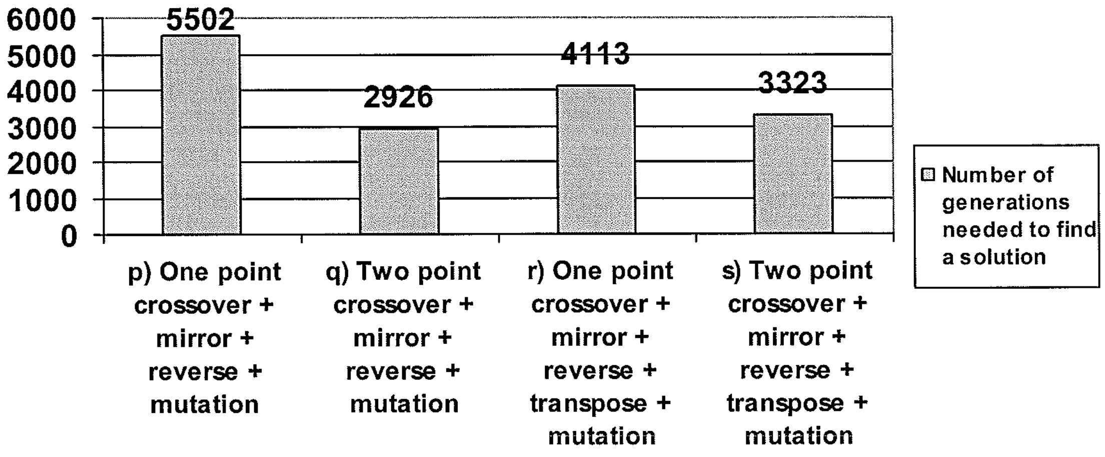
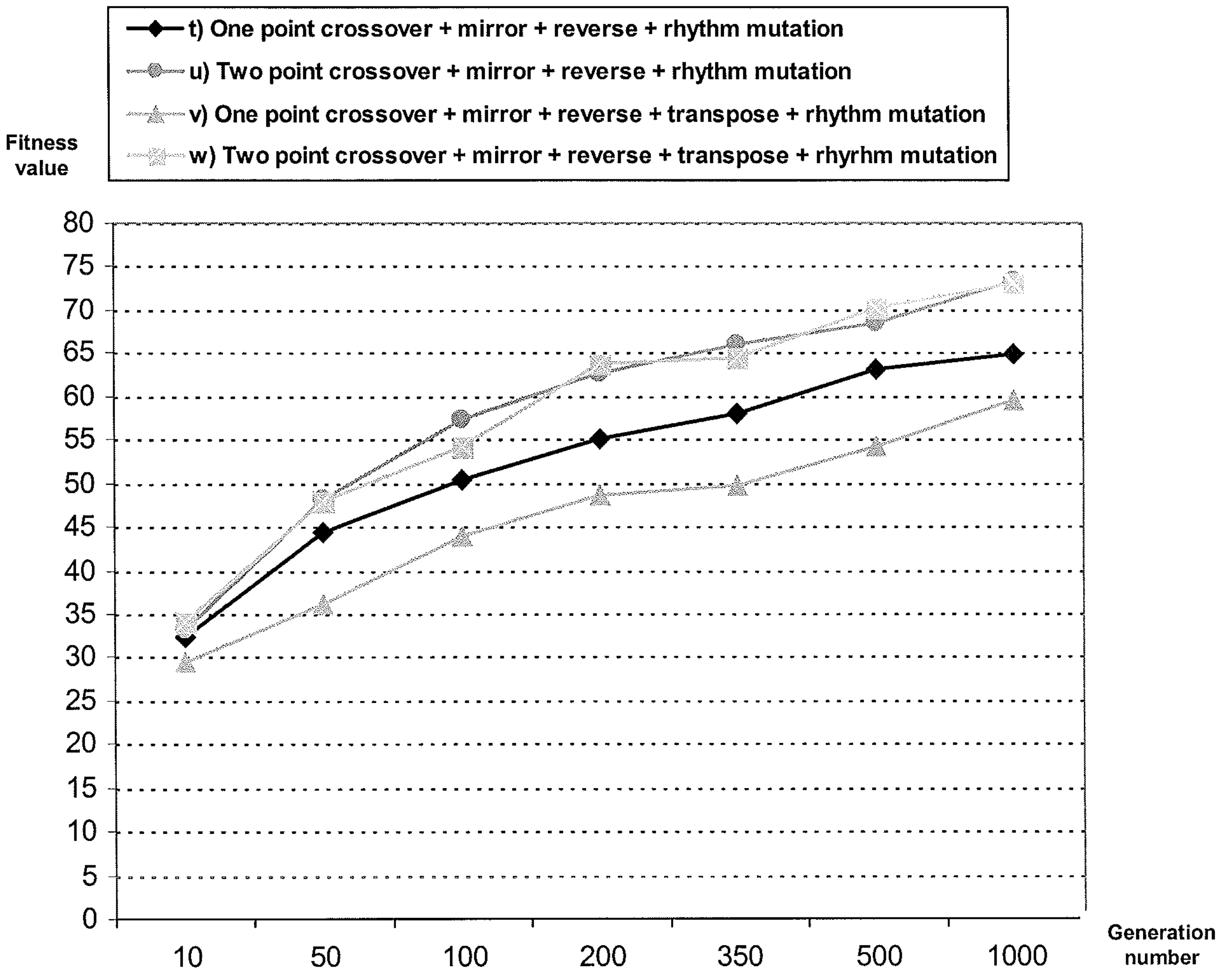


FIGURE [44]

Serial Application of Class 2 musical functions with One/two point crossover and rhythm mutation



GENERATION NUMBER / GENETIC OPERATION:	10	50	100	200	350	500	1000
t) One point crossover + mirror + reverse + rhythm mutation	32,4	44,4	50,4	55,0	58,0	63,0	64,8
u) Two point crossover + mirror + reverse + rhythm mutation	33,2	48,2	57,2	62,6	66,0	68,4	73,4
v) One point crossover + mirror + reverse + transpose + rhythm mutation	29,4	36,2	44,0	48,6	49,6	54,2	59,4
w) Two point crossover + mirror + reverse + transpose + rhythm mutation	34,2	47,8	54,2	63,8	64,4	70,2	73,2

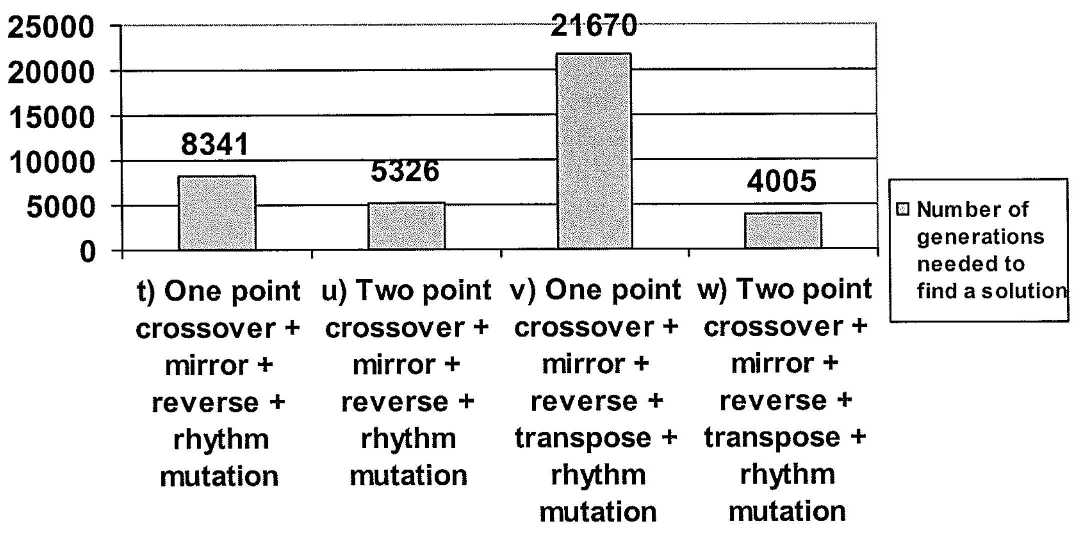
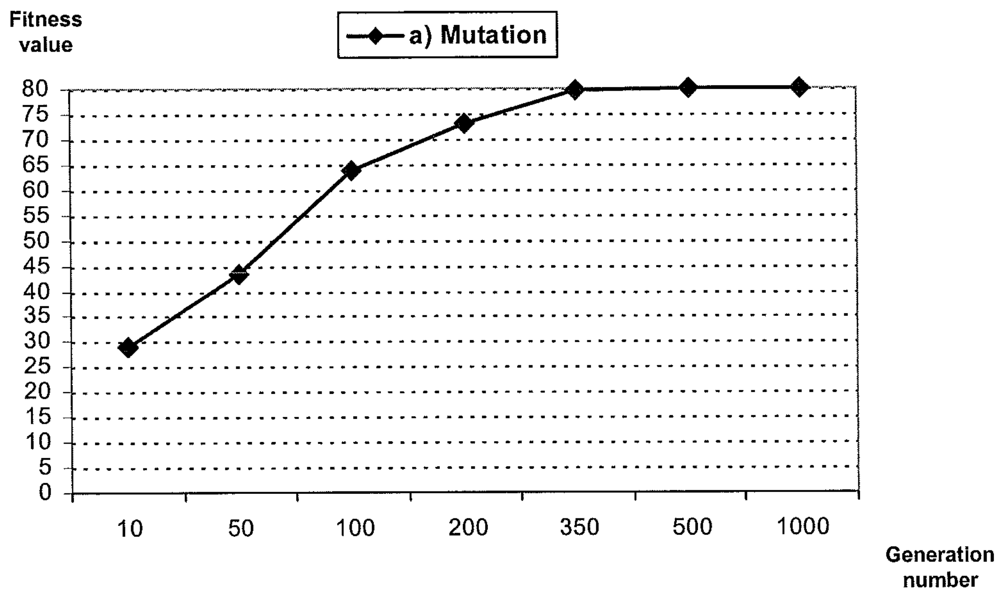


FIGURE [45]

- Mutation



GENETIC OPERATION: \ GENERATION NUMBER	10	50	100	200	350	500	1000
a) Mutation	29,0	43,2	63,8	73,2	79,6	80,0	80,0

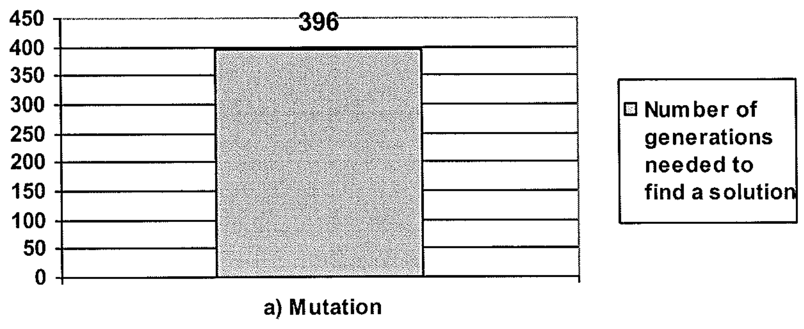


FIGURE [46]

7. Conclusions:

In the next table the best genetic operations are presented:

Position	Class	Generation needed to find a solution
1	One point crossover + mutation (Class 1)	329
2	One point crossover + mirror + mutation (Class 2)	377
3	Mutation (Class 1)	396
4	Two point crossover + mutation (Class 1)	509
5	Two point crossover + mirror + mutation (Class 2)	539
6	Two point crossover + transpose + mutation (Class 2)	656
7	One point crossover + transpose + rhythm mutation (Class 2)	730

FIGURE [47]

The class 1 (see figure 41) provides optimum results since they are three of the four best values (see figure 47). The results of this class are better than those of the class two (see figures from 42 to 45) except for One point crossover + mirror + mutation, which provides the second best value.

Although before testing the program it was expected that the joint use of several musical functions would provide better results than the use of each one separately, it is shown that the classical GA performed better.

One point crossover - Two point crossover.

In the figures 44 and 45 diagrams, the two point crossover is always better than the one point crossover, although the values are not good. If the figures 42 and 43 graphics are compared, it is difficult to get an accurate conclusion. This is due to fact that the results depend on the musical function which is applied (reverse, transpose or mirror).

Reverse – Transpose – Mirror.

The reverse musical function presents really poor values, regardless of the operation it is mixed with. This is the main reason why the results obtained when different musical functions are used at the same time are worse than expected (see figures 44 and 45). The transpose function gives acceptable results in all cases, especially when is combined with classical mutation. The mirror function provides very good values when is mixed with mutation. When this function is mixed with rhythm mutation, the fitness values are also acceptable although worse than the classic mutation (see figures 42 and 43).

Mutation – Rhythm Mutation.

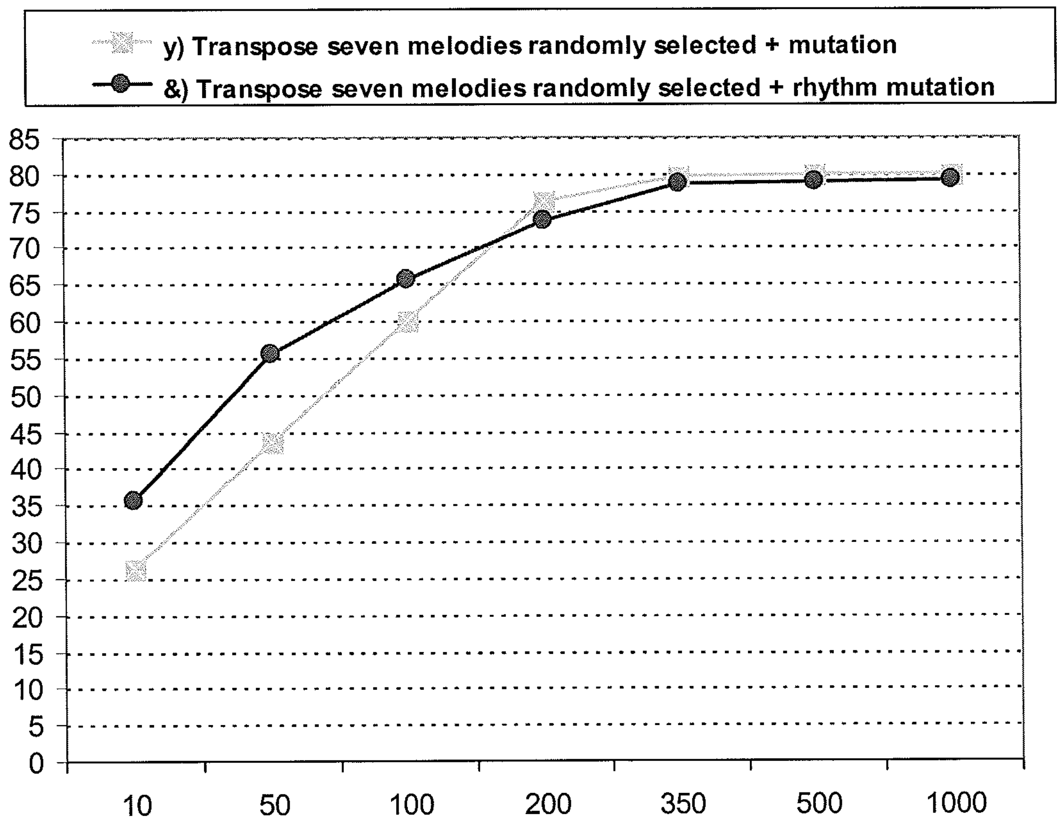
The mutation is always better than the rhythm mutation except for the figure 43 graphic (One point crossover mixed with only one musical function). This could be explained because of the optimal results that the mutation provides when it is used in isolation (see figure 46).

In further works it would be interesting to test the classic operations combined with various musical functions but without the reverse function; because it is possible that the poor results got in the figures 44 and 45 are due to the use of this function.

Appendix I

As it was previously mentioned, a third class is included in the program, although its study is not the aim of this work. Nevertheless, the user can experiment with these musical functions.

With regard to this class, the first column functions of the menu, supply very bad results because they are not combined with the mutation operation. As an interesting datum, the function: "Transpose seven melodies randomly selected" within this class, produced very good results. Thus, the following graphics have been included.



GENETIC OPERATION: \ GENERATION NUMBER	10	50	100	200	350	500	1000
y) Transpose seven melodies randomly selected + mutation	26,2	43,6	60,0	76,2	79,8	80,0	80,0
&) Transpose seven melodies randomly selected + rhythm mutation	35,6	55,6	65,6	73,6	78,6	79,0	79,2

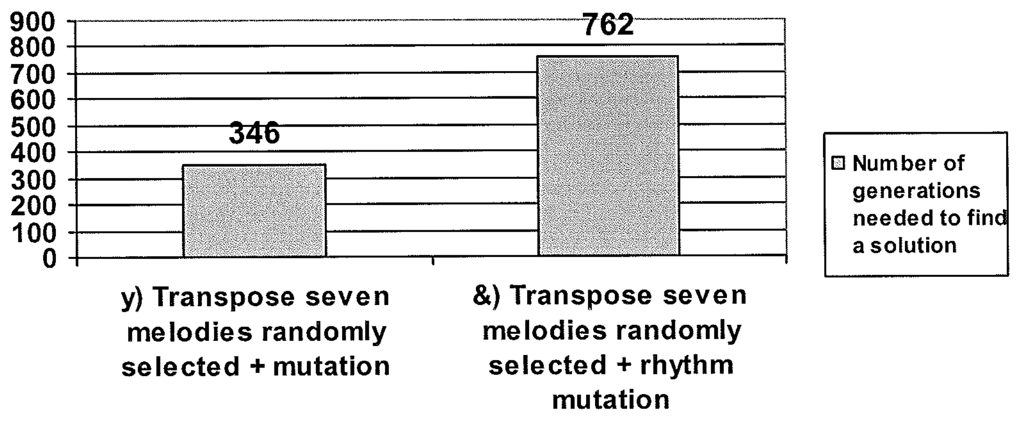

900
800
700
600
500
400
300
200
100
0
346
762
y) Transpose seven melodies randomly selected + mutation
&) Transpose seven melodies randomly selected + rhythm mutation
▨ Number of generations needed to find a solution

FIGURE [48]

Appendix II: Pascal code

```
Program Final;
uses crt;
const
     maxpop=100;
     maxstr=80;
     maxmelod=7;
type
    melodies= array [1..maxstr] of integer;
    population= array [1..maxpop] of melodies;
    fitness=array [1..maxmelod] of melodies;
    new=array [1..maxmelod] of integer;
    fit_value = array [1..maxpop] of new;
    mel_pos = array [1..2] of integer;
    allmel_pos = array [1..maxpop] of mel_pos;
const
     mel1:melodies=(2,14,10,14,10,14,10,14,10,14,6,14,4,14,14,14,8,14,10,14,
                    10,14,6,14,4,14,14,14,2,14,14,14,6,14,2,14,4,14,6,14,8,
                    14,6,14,4,14,14,14,6,14,2,14,4,14,6,14,8,14,6,14,4,14,14,
                    14,2,14,10,14,10,14,6,14,4,14,14,14,2,14,14,14);

     mel2:melodies=(2,14,10,14,10,14,10,14,10,8,6,14,4,14,14,14,8,14,10,14,
                   10,14,8,14,6,14,14,14,4,14,14,14,6,14,2,14,2,14,2,14,4,
                   6,8,6,4,14,14,14,6,14,2,14,2,14,2,14,4,6,8,6,4,14,14,14,
                    2,14,10,14,10,8,6,14,4,14,14,14,2,14,14,14);

     mel3:melodies=(2,14,2,14,10,14,10,14,10,14,10,14,8,6,4,14,6,14,8,14,10,
                   14,10,14,8,6,4,14,14,14,2,14,14,14,6,14,4,2,4,14,4,6,8,
                    14,8,6,4,14,14,14,6,14,4,2,4,14,4,6,8,14,8,6,4,14,14,14,
                   2,14,2,10,10,14,8,6,4,14,14,14,2,14);

     mel4:melodies=(2,14,10,14,10,14,10,14,10,14,10,14,8,6,4,14,6,14,8,14,10,
                   14,10,14,8,6,4,14,14,14,2,14,14,14,6,14,2,14,4,14,4,6,8,
                   14,6,14,4,14,14,14,6,14,2,14,4,14,4,6,8,14,6,14,4,14,12,2,
                    2,14,10,14,10,14,6,14,4,14,14,14,2,14);

     mel5:melodies=(10,14,14,14,10,14,10,14,10,14,6,14,4,14,14,14,4,14,6,14,8,
                  14,10,14,6,14,6,14,4,14,14,14,2,14,6,14,4,14,6,14,8,14,6,
                  14,4,14,14,14,2,14,14,14,4,14,6,14,8,14,6,14,4,14,14,14,2,
                   14,10,14,10,14,6,10,4,14,14,14,2,14,14,14);

     mel6:melodies=(10,14,14,14,10,14,10,14,10,14,6,14,4,14,14,14,4,14,6,14,8,
                  14,10,14,6,14,6,14,4,14,14,14,6,14,2,14,0,14,0,2,4,14,6,14,
                  4,14,14,14,2,14,6,14,4,14,6,14,8,14,6,14,4,14,4,14,2,14,10,
                  14,10,14,6,14,4,14,14,14,2,14,14,14);

     mel7:melodies=(2,14,14,14,10,14,14,14,10,14,8,6,4,14,2,14,2,14,4,14,4,2,2,
                  14,4,14,14,14,2,14,14,14,6,14,2,14,4,14,14,14,6,14,2,14,4,
                  14,14,8,6,14,4,2,4,14,14,8,6,14,4,2,4,14,14,4,8,14,10,14,10,
                     8,6,14,4,14,14,14,2,14,14,14);

 var
   f_function: fitness;
   pop1,pop2: population;
   cont,proc,gener,i:integer;
   opt1,opt2,opt3,class,wait:char;
   fin,punt: allmel_pos;
   value: fit_value;
   label 1;
   label 2;
```

```
{-------------fill the population melodies------------------}
procedure fill_fitness(var ffunction:fitness);
begin
     f_function[1]:=mel1;
     f_function[2]:=mel2;
     f_function[3]:=mel3;
     f_function[4]:=mel4;
     f_function[5]:=mel5;
     f_function[6]:=mel6;
     f_function[7]:=mel7;
end;

{-----------Random generation of the population)---------}
procedure Generate_pop(var pop:population);
var
   i,j:integer;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          for j:=1 to maxstr do
          begin
               pop[i,j]:=random(15);
               while pop[i,j] mod 2 <> 0 do
               begin
                    pop[i,j]:= random(15);
               end;
          end;
     end;
end;
{---Comparison note to note with every melody of the fitness function---}
procedure note_to_note(var pop:population;
                       var ffunction: fitness;
                       var ff_value: fit_value);
var
   i,j,k:integer;
begin
     for i:=1 to maxpop do
     begin
          for k:=1 to maxmelod do
          begin
               ff_value[i,k]:=0;
               for j:=1 to maxstr do
               begin
                    if pop[i,j]=ffunction[k,j] then
                    begin
                         ff_value[i,k]:=ff_value[i,k]+1;
                    end;
               end;
          end;
     end;
end;

{------Save the best fitness value and its position----}
procedure save_pos_mel (var ff_value: fit_value;
                        var pun: allmel_pos);
var
   i,j:integer;
begin
     for i:=1 to maxpop do
     begin
          pun[i,2]:=0;
          for j:=1 to maxmelod do
          begin
               if ff_value[i,j] > pun[i,2] then
               begin
                    pun[i,2]:=ff_value[i,j];
                    pun[i,1]:=j;
               end;
          end;
     end;
end;

{----------take the 50 best melodies and copy to pop2--------------}
procedure bestt(var pun: allmel_pos;
                  var pop,popp: population);
var
   i,j,max:integer;
   pun_aux: allmel_pos;
begin
     pun_aux:=pun;
     for i:=1 to maxpop div 2 do
     begin
          max:=1;
          for j:=1 to maxpop do
          begin
               if pun[max,2] < pun[j,2] then
               begin
                    max:=j;
               end
          end;
          popp[i]:=pop[max];
          pun[j,max]:=0;
     end;
     pun:=pun_aux;
end;
{-----take the 50 best melodies and copy to the second part of pop2----}
procedure bestt2(var pun: allmel_pos;
                  var pop,popp: population);
var
   i,j,max:integer;
   pun_aux: allmel_pos;
begin
     pun_aux:=pun;
     for i:=maxpop div 2 +1 to maxpop do
     begin
          max:=1;
          for j:=1 to maxpop do
          begin
               if pun[max,2] < pun[j,2] then
               begin
                    max:=j;
               end
          end;
          popp[i]:=pop[max];
          pun[j,max]:=0;
     end;
     pun:=pun_aux;
end;

{---------------------shows punctuation-----------------------}
procedure shows_punct(var pun: allmel_pos);
var
   i:integer;
begin
     for i:=1 to maxpop do
     begin
          write(pun[i,2]);
          if (i=20) or (i=40) or (i=60) or (i=80) or (i=100) or (i=120) or
          (i=140) or (i=160) or (i=180) then
          begin
               writeln;
          end
          else
               write(',');
     end;
end;

{---------Copy the best 50 melodies from pop2 to pop1------------}
procedure copy1(var pop,popp:population);
var
   i:Integer;
begin
     for i:=1 to maxpop div 2 do
     begin
          pop[i]:=popp[i];
     end;
end;
{-------------------generate 50 new random melodies---------------}
procedure Generate2(var pop:population);
var
   i,j:integer;
begin
     randomize;
     for i:=maxpop div 2 +1 to maxpop do
     begin
          for j:=1 to maxstr do
          begin
               pop[i,j]:=random(15);
               while pop[i,j] mod 2 <> 0 do
               begin
                    pop[i,j]:= random(15);
               end;
          end;
     end;
end;

{--------Copy the pop2 in pop1------------}
procedure copy2(var pop,popp:population);
var
   i:Integer;
begin
     for i:=1 to maxpop do
     begin
          pop[i]:=popp[i];
     end;
end;

{--------------Class1: one point crossover--------------------}
procedure one_crossover(var pop:population);
var
   i,j,aux,pos:integer;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
```

```
                    pos:=random(maxstr);
               end;
               for j:=pos to maxstr do
               begin
                    aux:= pop[i,j];
                    pop[i,j]:= pop[i+maxpop div 2,j];
                    pop[i+maxpop div 2,j]:=aux;
               end;
          end;
end;

{-----------------Class1: two-point crossover-------------------------}
procedure two_crossover(var pop: population);
var
    i,j,aux,pos1,pos2:integer;
begin
          randomize;
          for i:=1 to maxpop div 2 do
          begin
               pos1:=0;
               while pos1 = 0 do
               begin
                    pos1:=random(maxstr-1);
               end;
               pos2:=0;
               while (pos2=0) or (pos2<=pos1+1) do
               begin
                    pos2:=random(maxstr+1);
               end;
               for j:=pos1 to pos2 do
               begin
                    aux:= pop[i,j];
                    pop[i,j]:= pop[i+maxpop div 2,j];
                    pop[i+maxpop div 2,j]:=aux;
               end;
          end;
end;
{-----------One point crossover + transpose----------------}
procedure one_crossover_transp(var pop:population);
var
    trans,mirr,i,j,k,l,aux,pos:integer;
    aux2:melodies;
begin
          randomize;
          for i:=1 to maxpop div 2 do
          begin
               pos:=0;
               while pos=0 do
               begin
                    pos:=random(maxstr);
               end;
               for j:=pos to maxstr do
               begin
                    aux:= pop[i,j];
                    pop[i,j]:= pop[i+maxpop div 2,j];
                    pop[i+maxpop div 2,j]:=aux;
               end;
               trans:=random(7);
               trans:=trans*2;
               for j:=pos to maxstr do
               begin
                    if pop[i,j] <= 10 then
                    begin
                    pop[i,j]:=pop[i,j]+trans;
                    if pop[i,j] > 10 then
                    begin
                         pop[i,j]:=pop[i,j]-12;
                    end;
               end;
          end;
          trans:=random(7);
          trans:=trans*2;
          for j:=pos to maxstr do
          begin
               if pop[i+maxpop div 2,j] <= 10 then
               begin
                    pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]+trans;
                    if pop[i+maxpop div 2,j] > 10 then
                    begin
                         pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]-12;
                    end;
               end;
          end;
     end;
end;

{----------One point crossover + reverse-------------}
procedure one_crossover_rever(var pop:population);
var
   mirr,i,j,k,l,aux,pos:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr);
          end;
          for j:=pos to maxstr do
          begin
               aux:= pop[i,j];
               pop[i,j]:= pop[i+maxpop div 2,j];
               pop[i+maxpop div 2,j]:=aux;
          end;
          k:=1;
          for j:=maxstr downto pos do
          begin
               aux2[k]:=pop[i,j];
               k:=k+1;
          end;
          k:=1;
          for l:=pos to maxstr do
          begin
               pop[i,l]:=aux2[k];
               k:=k+1;
          end;
          k:=1;
          for j:=maxstr downto pos do
          begin
               aux2[k]:=pop[i+maxpop div 2,j];
               k:=k+1;
          end;
          k:=1;
          for l:=pos to maxstr do
          begin
               pop[i+maxpop div 2,l]:=aux2[k];
               k:=k+1;
          end;
     end;
end;

{-----------One point crossover + mirror -------------}
procedure one_crossover_mirr(var pop:population);
var
   mirr,i,j,k,l,aux,pos:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr);
          end;
          for j:=pos to maxstr do
          begin
               aux:= pop[i,j];
               pop[i,j]:= pop[i+maxpop div 2,j];
               pop[i+maxpop div 2,j]:=aux;
          end;
          mirr:=pop[i,pos];
          for j:=pos to maxstr do
          begin
               if (mirr <> 12) and (mirr <> 14) then
               begin
                    if (mirr = 0) or (mirr = 6) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=0;
                              2: pop[i,j]:=10;
                              4: pop[i,j]:=8;
                              6: pop[i,j]:=6;
                              8: pop[i,j]:=4;
                              10: pop[i,j]:=2;
                         end;
                    end;
                    if (mirr = 2) or (mirr = 8) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=4;
                              2: pop[i,j]:=2;
                              4: pop[i,j]:=0;
                              6: pop[i,j]:=10;
                              8: pop[i,j]:=8;
                              10: pop[i,j]:=6;
                         end;
                    end;
                    if (mirr = 4) or (mirr = 10) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=8;
                              2: pop[i,j]:=6;
                              4: pop[i,j]:=4;
                              6: pop[i,j]:=2;
                              8: pop[i,j]:=0;
                              10: pop[i,j]:=10;
                         end;
                    end;
               end;
          end;
          mirr:=pop[i+maxpop div 2,pos];
          for j:=pos to maxstr do
          begin
               if (mirr <> 12) and (mirr <> 14) then
               begin
                    if (mirr = 0) or (mirr = 6) then
                    begin
                         case pop[i+maxpop div 2,j] of
                              0: pop[i+maxpop div 2,j]:=0;
                              2: pop[i+maxpop div 2,j]:=10;
                              4: pop[i+maxpop div 2,j]:=8;
                              6: pop[i+maxpop div 2,j]:=6;
                              8: pop[i+maxpop div 2,j]:=4;
                              10: pop[i+maxpop div 2,j]:=2;
                         end;
                    end;
                    if (mirr = 2) or (mirr = 8) then
                    begin
                         case pop[i,j] of
                              0: pop[i+maxpop div 2,j]:=4;
                              2: pop[i+maxpop div 2,j]:=2;
                              4: pop[i+maxpop div 2,j]:=0;
                              6: pop[i+maxpop div 2,j]:=10;
                              8: pop[i+maxpop div 2,j]:=8;
                              10: pop[i+maxpop div 2,j]:=6;
                         end;
                    end;
                    if (mirr = 4) or (mirr = 10) then
                    begin
                         case pop[i+maxpop div 2,j] of
                              0: pop[i+maxpop div 2,j]:=8;
                              2: pop[i+maxpop div 2,j]:=6;
                              4: pop[i+maxpop div 2,j]:=4;
                              6: pop[i+maxpop div 2,j]:=2;
                              8: pop[i+maxpop div 2,j]:=0;
                              10: pop[i+maxpop div 2,j]:=10;
                         end;
                    end;
               end;
          end;
     end;
end;

{-----------One point crossover + mirror and reverse--------------}
procedure one_crossover_mirr_rever(var pop:population);
var
   mirr,i,j,k,l,aux,pos:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr);
          end;
          for j:=pos to maxstr do
          begin
          aux:= pop[i,j];
          pop[i,j]:= pop[i+maxpop div 2,j];
          pop[i+maxpop div 2,j]:=aux;
     end;
     mirr:=pop[i,pos];
     for j:=pos to maxstr do
     begin
          if (mirr <> 12) and (mirr <> 14) then
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=0;
                         2: pop[i,j]:=10;
                         4: pop[i,j]:=8;
                         6: pop[i,j]:=6;
                         8: pop[i,j]:=4;
                         10: pop[i,j]:=2;
                    end;
               end;
               if (mirr = 2) or (mirr = 8) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=4;
                         2: pop[i,j]:=2;
                         4: pop[i,j]:=0;
                         6: pop[i,j]:=10;
                         8: pop[i,j]:=8;
                         10: pop[i,j]:=6;
                    end;
               end;
               if (mirr = 4) or (mirr = 10) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=8;
                         2: pop[i,j]:=6;
                         4: pop[i,j]:=4;
                         6: pop[i,j]:=2;
                         8: pop[i,j]:=0;
                         10: pop[i,j]:=10;
                    end;
               end;
          end;
     end;
     k:=1;
     for j:=maxstr downto pos do
     begin
          aux2[k]:=pop[i,j];
          k:=k+1;
     end;
     k:=1;
     for l:=pos to maxstr do
     begin
          pop[i,l]:=aux2[k];
          k:=k+1;
     end;
     mirr:=pop[i+maxpop div 2,pos];
     for j:=pos to maxstr do
     begin
          if (mirr <> 12) and (mirr <> 14) then
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
```

```
                              case pop[i+maxpop div 2,j] of
                                   0: pop[i+maxpop div 2,j]:=0;
                                   2: pop[i+maxpop div 2,j]:=10;
                                   4: pop[i+maxpop div 2,j]:=8;
                                   6: pop[i+maxpop div 2,j]:=6;
                                   8: pop[i+maxpop div 2,j]:=4;
                                   10: pop[i+maxpop div 2,j]:=2;
                              end;
                         end;
                         if (mirr = 2) or (mirr = 8) then
                         begin
                              case pop[i,j] of
                                   0: pop[i+maxpop div 2,j]:=4;
                                   2: pop[i+maxpop div 2,j]:=2;
                                   4: pop[i+maxpop div 2,j]:=0;
                                   6: pop[i+maxpop div 2,j]:=10;
                                   8: pop[i+maxpop div 2,j]:=8;
                                   10: pop[i+maxpop div 2,j]:=6;
                              end;
                         end;
                         if (mirr = 4) or (mirr = 10) then
                         begin
                              case pop[i+maxpop div 2,j] of
                                   0: pop[i+maxpop div 2,j]:=8;
                                   2: pop[i+maxpop div 2,j]:=6;
                                   4: pop[i+maxpop div 2,j]:=4;
                                   6: pop[i+maxpop div 2,j]:=2;
                                   8: pop[i+maxpop div 2,j]:=0;
                                   10: pop[i+maxpop div 2,j]:=10;
                              end;
                         end;
                    end;
               end;
               k:=1;
               for j:=maxstr downto pos do
               begin
                    aux2[k]:=pop[i+maxpop div 2,j];
                    k:=k+1;
               end;
               k:=1;
               for l:=pos to maxstr do
               begin
                    pop[i+maxpop div 2,l]:=aux2[k];
                    k:=k+1;
               end;
     end;
end;

{-------One point crossover + mirror,reverse and transpose-------}
procedure one_crossover_mirr_rever_transp(var pop:population);
var
   trans,mirr,i,j,k,l,aux,pos:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr);
          end;
          for j:=pos to maxstr do
```

```
begin
     aux:= pop[i,j];
     pop[i,j]:= pop[i+maxpop div 2,j];
     pop[i+maxpop div 2,j]:=aux;
end;
mirr:=pop[i,pos];
for j:=pos to maxstr do
begin
     if (mirr <> 12) and (mirr <> 14) then
     begin
          if (mirr = 0) or (mirr = 6) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=0;
                    2: pop[i,j]:=10;
                    4: pop[i,j]:=8;
                    6: pop[i,j]:=6;
                    8: pop[i,j]:=4;
                    10: pop[i,j]:=2;
               end;
          end;
          if (mirr = 2) or (mirr = 8) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=4;
                    2: pop[i,j]:=2;
                    4: pop[i,j]:=0;
                    6: pop[i,j]:=10;
                    8: pop[i,j]:=8;
                    10: pop[i,j]:=6;
               end;
          end;
          if (mirr = 4) or (mirr = 10) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=8;
                    2: pop[i,j]:=6;
                    4: pop[i,j]:=4;
                    6: pop[i,j]:=2;
                    8: pop[i,j]:=0;
                    10: pop[i,j]:=10;
               end;
          end;
     end;
end;
k:=1;
for j:=maxstr downto pos do
begin
     aux2[k]:=pop[i,j];
     k:=k+1;
end;
k:=1;
for l:=pos to maxstr do
begin
     pop[i,l]:=aux2[k];
     k:=k+1;
end;
trans:=random(7);
trans:=trans*2;
for j:=pos to maxstr do
begin
     if pop[i,j] <= 10 then
     begin
```

```
          pop[i,j]:=pop[i,j]+trans;
          if pop[i,j] > 10 then
          begin
               pop[i,j]:=pop[i,j]-12;
          end;
     end;
end;
mirr:=pop[i+maxpop div 2,pos];
for j:=pos to maxstr do
begin
     if (mirr <> 12) and (mirr <> 14) then
     begin
          if (mirr = 0) or (mirr = 6) then
          begin
               case pop[i+maxpop div 2,j] of
                    0: pop[i+maxpop div 2,j]:=0;
                    2: pop[i+maxpop div 2,j]:=10;
                    4: pop[i+maxpop div 2,j]:=8;
                    6: pop[i+maxpop div 2,j]:=6;
                    8: pop[i+maxpop div 2,j]:=4;
                    10: pop[i+maxpop div 2,j]:=2;
               end;
          end;
          if (mirr = 2) or (mirr = 8) then
          begin
               case pop[i,j] of
                    0: pop[i+maxpop div 2,j]:=4;
                    2: pop[i+maxpop div 2,j]:=2;
                    4: pop[i+maxpop div 2,j]:=0;
                    6: pop[i+maxpop div 2,j]:=10;
                    8: pop[i+maxpop div 2,j]:=8;
                    10: pop[i+maxpop div 2,j]:=6;
               end;
          end;
          if (mirr = 4) or (mirr = 10) then
          begin
               case pop[i+maxpop div 2,j] of
                    0: pop[i+maxpop div 2,j]:=8;
                    2: pop[i+maxpop div 2,j]:=6;
                    4: pop[i+maxpop div 2,j]:=4;
                    6: pop[i+maxpop div 2,j]:=2;
                    8: pop[i+maxpop div 2,j]:=0;
                    10: pop[i+maxpop div 2,j]:=10;
               end;
          end;
     end;
end;
k:=1;
for j:=maxstr downto pos do
begin
     aux2[k]:=pop[i+maxpop div 2,j];
     k:=k+1;
end;
k:=1;
for l:=pos to maxstr do
begin
     pop[i+maxpop div 2,l]:=aux2[k];
     k:=k+1;
end;
trans:=random(7);
trans:=trans*2;
for j:=pos to maxstr do
begin
```

```
                if pop[i+maxpop div 2,j] <= 10 then
                begin
                     pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]+trans;
                     if pop[i+maxpop div 2,j] > 10 then
                     begin
                          pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]-12;
                     end;
                end;
           end;
      end;
end;

{------------ two point crossover + transpose ------------}
procedure two_crossover_transp(var pop:population);
var
   trans,mirr,i,j,k,l,aux,pos,pos2:integer;
   aux2:melodies;
begin
      randomize;
      for i:=1 to maxpop div 2 do
      begin
           pos:=0;
           while pos=0 do
           begin
                pos:=random(maxstr-1);
           end;
           pos2:=0;
           while (pos2=0) or (pos2<=pos+1) do
           begin
                pos2:=random(maxstr+1);
           end;
           for j:=pos to pos2 do
           begin
                aux:= pop[i,j];
                pop[i,j]:= pop[i+maxpop div 2,j];
                pop[i+maxpop div 2,j]:=aux;
           end;
           trans:=random(7);
           trans:=trans*2;
           for j:=pos to pos2 do
           begin
                if pop[i,j] <= 10 then
                begin
                     pop[i,j]:=pop[i,j]+trans;
                     if pop[i,j] > 10 then
                     begin
                          pop[i,j]:=pop[i,j]-12;
                     end;
                end;
           end;
           trans:=random(7);
           trans:=trans*2;
           for j:=pos to pos2 do
           begin
                if pop[i+maxpop div 2,j] <= 10 then
                begin
                     pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]+trans;
                     if pop[i+maxpop div 2,j] > 10 then
                     begin
                          pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]-12;
                     end;
                end;
           end;
```

```
     end;
end;

{----------------two point croosover + reverse-------------------}
procedure two_crossover_rever(var pop:population);
var
   mirr,i,j,k,l,aux,pos,pos2:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          for j:=pos to pos2 do
          begin
               aux:= pop[i,j];
               pop[i,j]:= pop[i+maxpop div 2,j];
               pop[i+maxpop div 2,j]:=aux;
          end;
          k:=1;
          for j:=pos2 downto pos do
          begin
               aux2[k]:=pop[i,j];
               k:=k+1;
          end;
          k:=1;
          for l:=pos to pos2 do
          begin
               pop[i,l]:=aux2[k];
               k:=k+1;
          end;
          k:=1;
          for j:=pos2 downto pos do
          begin
               aux2[k]:=pop[i+maxpop div 2,j];
               k:=k+1;
          end;
          k:=1;
          for l:=pos to pos2 do
          begin
               pop[i+maxpop div 2,l]:=aux2[k];
               k:=k+1;
          end;
     end;
end;

{----------------two point crossover + mirror--------------------}
procedure two_crossover_mirr(var pop:population);
var
   mirr,i,j,k,l,aux,pos,pos2:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
```

```
begin
     pos:=0;
     while pos=0 do
     begin
          pos:=random(maxstr-1);
     end;
     pos2:=0;
     while (pos2=0) or (pos2<=pos+1) do
     begin
          pos2:=random(maxstr+1);
     end;
     for j:=pos to pos2 do
     begin
          aux:= pop[i,j];
          pop[i,j]:= pop[i+maxpop div 2,j];
          pop[i+maxpop div 2,j]:=aux;
     end;
     mirr:=pop[i,pos];
     for j:=pos to pos2 do
     begin
          if (mirr <> 12) and (mirr <> 14) then
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=0;
                         2: pop[i,j]:=10;
                         4: pop[i,j]:=8;
                         6: pop[i,j]:=6;
                         8: pop[i,j]:=4;
                         10: pop[i,j]:=2;
                    end;
               end;
               if (mirr = 2) or (mirr = 8) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=4;
                         2: pop[i,j]:=2;
                         4: pop[i,j]:=0;
                         6: pop[i,j]:=10;
                         8: pop[i,j]:=8;
                         10: pop[i,j]:=6;
                    end;
               end;
               if (mirr = 4) or (mirr = 10) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=8;
                         2: pop[i,j]:=6;
                         4: pop[i,j]:=4;
                         6: pop[i,j]:=2;
                         8: pop[i,j]:=0;
                         10: pop[i,j]:=10;
                    end;
               end;
          end;
     end;
     mirr:=pop[i+maxpop div 2,pos];
     for j:=pos to pos2 do
     begin
          if (mirr <> 12) and (mirr <> 14) then
          begin
               if (mirr = 0) or (mirr = 6) then
```

```
                    begin
                         case pop[i+maxpop div 2,j] of
                              0: pop[i+maxpop div 2,j]:=0;
                              2: pop[i+maxpop div 2,j]:=10;
                              4: pop[i+maxpop div 2,j]:=8;
                              6: pop[i+maxpop div 2,j]:=6;
                              8: pop[i+maxpop div 2,j]:=4;
                              10: pop[i+maxpop div 2,j]:=2;
                         end;
                    end;
                    if (mirr = 2) or (mirr = 8) then
                    begin
                         case pop[i,j] of
                              0: pop[i+maxpop div 2,j]:=4;
                              2: pop[i+maxpop div 2,j]:=2;
                              4: pop[i+maxpop div 2,j]:=0;
                              6: pop[i+maxpop div 2,j]:=10;
                              8: pop[i+maxpop div 2,j]:=8;
                              10: pop[i+maxpop div 2,j]:=6;
                         end;
                    end;
                    if (mirr = 4) or (mirr = 10) then
                    begin
                         case pop[i+maxpop div 2,j] of
                              0: pop[i+maxpop div 2,j]:=8;
                              2: pop[i+maxpop div 2,j]:=6;
                              4: pop[i+maxpop div 2,j]:=4;
                              6: pop[i+maxpop div 2,j]:=2;
                              8: pop[i+maxpop div 2,j]:=0;
                              10: pop[i+maxpop div 2,j]:=10;
                         end;
                    end;
               end;
          end;
     end;
end;

{-----------two point crossover + mirror and reverse-------------}
procedure two_crossover_mirr_rever(var pop:population);
var
   mirr,i,j,k,l,aux,pos,pos2:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          for j:=pos to pos2 do
          begin
               aux:= pop[i,j];
               pop[i,j]:= pop[i+maxpop div 2,j];
               pop[i+maxpop div 2,j]:=aux;
          end;
          mirr:=pop[i,pos];
```

```
for j:=pos to pos2 do
begin
     if (mirr <> 12) and (mirr <> 14) then
     begin
          if (mirr = 0) or (mirr = 6) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=0;
                    2: pop[i,j]:=10;
                    4: pop[i,j]:=8;
                    6: pop[i,j]:=6;
                    8: pop[i,j]:=4;
                    10: pop[i,j]:=2;
               end;
          end;
          if (mirr = 2) or (mirr = 8) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=4;
                    2: pop[i,j]:=2;
                    4: pop[i,j]:=0;
                    6: pop[i,j]:=10;
                    8: pop[i,j]:=8;
                    10: pop[i,j]:=6;
               end;
          end;
          if (mirr = 4) or (mirr = 10) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=8;
                    2: pop[i,j]:=6;
                    4: pop[i,j]:=4;
                    6: pop[i,j]:=2;
                    8: pop[i,j]:=0;
                    10: pop[i,j]:=10;
               end;
          end;
     end;
end;
k:=1;
for j:=pos2 downto pos do
begin
     aux2[k]:=pop[i,j];
     k:=k+1;
end;
k:=1;
for l:=pos to pos2 do
begin
     pop[i,l]:=aux2[k];
     k:=k+1;
end;
mirr:=pop[i+maxpop div 2,pos];
for j:=pos to pos2 do
begin
     if (mirr <> 12) and (mirr <> 14) then
     begin
          if (mirr = 0) or (mirr = 6) then
          begin
               case pop[i+maxpop div 2,j] of
                    0: pop[i+maxpop div 2,j]:=0;
                    2: pop[i+maxpop div 2,j]:=10;
                    4: pop[i+maxpop div 2,j]:=8;
                    6: pop[i+maxpop div 2,j]:=6;
```

```
                        8: pop[i+maxpop div 2,j]:=4;
                        10: pop[i+maxpop div 2,j]:=2;
                   end;
              end;
              if (mirr = 2) or (mirr = 8) then
              begin
                   case pop[i,j] of
                        0: pop[i+maxpop div 2,j]:=4;
                        2: pop[i+maxpop div 2,j]:=2;
                        4: pop[i+maxpop div 2,j]:=0;
                        6: pop[i+maxpop div 2,j]:=10;
                        8: pop[i+maxpop div 2,j]:=8;
                        10: pop[i+maxpop div 2,j]:=6;
                   end;
              end;
              if (mirr = 4) or (mirr = 10) then
              begin
                   case pop[i+maxpop div 2,j] of
                        0: pop[i+maxpop div 2,j]:=8;
                        2: pop[i+maxpop div 2,j]:=6;
                        4: pop[i+maxpop div 2,j]:=4;
                        6: pop[i+maxpop div 2,j]:=2;
                        8: pop[i+maxpop div 2,j]:=0;
                        10: pop[i+maxpop div 2,j]:=10;
                   end;
              end;
         end;
    end;
    k:=1;
    for j:=pos2 downto pos do
    begin
         aux2[k]:=pop[i+maxpop div 2,j];
         k:=k+1;
    end;
    k:=1;
    for l:=pos to pos2 do
    begin
         pop[i+maxpop div 2,l]:=aux2[k];
         k:=k+1;
    end;
  end;
end;

{-------two point crossover + mirror, reverse and transpose-------}
procedure two_crossover_mirr_rever_transp(var pop:population);
var
   trans,mirr,i,j,k,l,aux,pos,pos2:integer;
   aux2:melodies;
begin
     randomize;
     for i:=1 to maxpop div 2 do
     begin
          pos:=0;
          while pos=0 do
          begin
               pos:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          for j:=pos to pos2 do
```

```
begin
     aux:= pop[i,j];
     pop[i,j]:= pop[i+maxpop div 2,j];
     pop[i+maxpop div 2,j]:=aux;
end;
mirr:=pop[i,pos];
for j:=pos to pos2 do
begin
     if (mirr <> 12) and (mirr <> 14) then
     begin
          if (mirr = 0) or (mirr = 6) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=0;
                    2: pop[i,j]:=10;
                    4: pop[i,j]:=8;
                    6: pop[i,j]:=6;
                    8: pop[i,j]:=4;
                    10: pop[i,j]:=2;
               end;
          end;
          if (mirr = 2) or (mirr = 8) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=4;
                    2: pop[i,j]:=2;
                    4: pop[i,j]:=0;
                    6: pop[i,j]:=10;
                    8: pop[i,j]:=8;
                    10: pop[i,j]:=6;
               end;
          end;
          if (mirr = 4) or (mirr = 10) then
          begin
               case pop[i,j] of
                    0: pop[i,j]:=8;
                    2: pop[i,j]:=6;
                    4: pop[i,j]:=4;
                    6: pop[i,j]:=2;
                    8: pop[i,j]:=0;
                    10: pop[i,j]:=10;
               end;
          end;
     end;

end;
k:=1;
for j:=pos2 downto pos do
begin
     aux2[k]:=pop[i,j];
     k:=k+1;
end;
k:=1;
for l:=pos to pos2 do
begin
     pop[i,l]:=aux2[k];
     k:=k+1;
end;
trans:=random(7);
trans:=trans*2;
for j:=pos to pos2 do
begin
     if pop[i,j] <= 10 then
```

```
          begin
               pop[i,j]:=pop[i,j]+trans;
               if pop[i,j] > 10 then
               begin
                    pop[i,j]:=pop[i,j]-12;
               end;
          end;
     end;
     mirr:=pop[i+maxpop div 2,pos];
     for j:=pos to pos2 do
     begin
          if (mirr <> 12) and (mirr <> 14) then
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
                    case pop[i+maxpop div 2,j] of
                         0: pop[i+maxpop div 2,j]:=0;
                         2: pop[i+maxpop div 2,j]:=10;
                         4: pop[i+maxpop div 2,j]:=8;
                         6: pop[i+maxpop div 2,j]:=6;
                         8: pop[i+maxpop div 2,j]:=4;
                         10: pop[i+maxpop div 2,j]:=2;
                    end;
               end;
               if (mirr = 2) or (mirr = 8) then
               begin
                    case pop[i,j] of
                         0: pop[i+maxpop div 2,j]:=4;
                         2: pop[i+maxpop div 2,j]:=2;
                         4: pop[i+maxpop div 2,j]:=0;
                         6: pop[i+maxpop div 2,j]:=10;
                         8: pop[i+maxpop div 2,j]:=8;
                         10: pop[i+maxpop div 2,j]:=6;
                    end;
               end;
               if (mirr = 4) or (mirr = 10) then
               begin
                    case pop[i+maxpop div 2,j] of
                         0: pop[i+maxpop div 2,j]:=8;
                         2: pop[i+maxpop div 2,j]:=6;
                         4: pop[i+maxpop div 2,j]:=4;
                         6: pop[i+maxpop div 2,j]:=2;
                         8: pop[i+maxpop div 2,j]:=0;
                         10: pop[i+maxpop div 2,j]:=10;
                    end;
               end;
          end;
     end;
     k:=1;
     for j:=pos2 downto pos do
     begin
          aux2[k]:=pop[i+maxpop div 2,j];
          k:=k+1;
     end;
     k:=1;
     for l:=pos to pos2 do
     begin
          pop[i+maxpop div 2,l]:=aux2[k];
          k:=k+1;
     end;
     trans:=random(7);
     trans:=trans*2;
     for j:=pos to pos2 do
```

```
          begin
               if pop[i+maxpop div 2,j] <= 10 then
               begin
                    pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]+trans;
                    if pop[i+maxpop div 2,j] > 10 then
                    begin
                         pop[i+maxpop div 2,j]:=pop[i+maxpop div 2,j]-12;
                    end;
               end;
          end;
     end;
end;

{---------------------- class1: simple mutation ------------------------}
procedure mutation (var pop:population);
var
   i,pos:integer;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          pos:=0;
          while pos = 0 do
          begin
               pos:=random(maxstr+1);
          end;
          pop[i,pos]:=random(15);
          while pop[i,pos] mod 2 <> 0 do
          begin
               pop[i,pos]:= random(15);
          end;
     end;
end;

{--------Mutation rhythm in every melodie of the population--------}
procedure Mutation_rhythm1(var pop:population);
var
   i,j:integer;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          j:=0;
          while j=0 do
          begin
               j:=random(maxstr-2);
          end;
          if (pop[i,j] <>14) and (pop[i,j+1] <> 14) and (pop[i,j+2] <> 14) then
          begin
               pop[i,j]:=14;
          end
          else
          begin
               pop[i,j]:= random(15);
               while pop[i,j] mod 2 <> 0 do
               begin
                    pop[i,j]:= random(15);
               end;
          end;
     end;
end;

{----Mutation rhythm2: Take a random bar of every melody and convert----
```

```
--in minim the last note ( in randomly selected melodies of the population--}
procedure Mutation_rhythm2(var pop:population);
var
   i,bar:integer;
begin
     randomize;
     i:=0;
     while i =0 do
     begin
          i:=random(12);
     end;
     while i < maxpop do
     begin
          bar:=random(10);
          bar:=bar*8+1;
          pop[i,bar+5]:=14;
          pop[i,bar+6]:=14;
          pop[i,bar+7]:=14;
     i:=i+random(12);
     end;
end;

{----------Mirror taking the first note like base----------
 ------------in every melody of the population-------------}
procedure mirror_first (var pop:population);
var
   i,j,mirr:integer;
begin
     for i:=1 to maxpop do
     begin
          mirr:=pop[i,1];
          for j:=1 to maxstr do
          begin
               if (mirr <> 12) and (mirr <> 14) then
               begin
                    if (mirr = 0) or (mirr = 6) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=0;
                              2: pop[i,j]:=10;
                              4: pop[i,j]:=8;
                              6: pop[i,j]:=6;
                              8: pop[i,j]:=4;
                              10: pop[i,j]:=2;
                         end;
                    end;
                    if (mirr = 2) or (mirr = 8) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=4;
                              2: pop[i,j]:=2;
                              4: pop[i,j]:=0;
                              6: pop[i,j]:=10;
                              8: pop[i,j]:=8;
                              10: pop[i,j]:=6;
                         end;
                    end;
                    if (mirr = 4) or (mirr = 10) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=8;
                              2: pop[i,j]:=6;
                              4: pop[i,j]:=4;
                              6: pop[i,j]:=2;
                              8: pop[i,j]:=0;
                              10: pop[i,j]:=10;
                         end;
                    end;
               end;
          end;
     end;
end;

{----------Mirror taking a random note like base-----------
--------------in every of the population----------------}
procedure mirror_random (var pop:population);
var
   i,j,mirr:integer;
begin
     for i:=1 to maxpop do
     begin
          mirr:=random(11);
          while mirr mod 2 <> 0 do
          begin
               mirr:= random(11);
          end;
          for j:=1 to maxstr do
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=0;
                         2: pop[i,j]:=10;
                         4: pop[i,j]:=8;
                         6: pop[i,j]:=6;
                         8: pop[i,j]:=4;
                         10: pop[i,j]:=2;
                    end;
               end
               else
               if (mirr = 2) or (mirr = 8) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=4;
                         2: pop[i,j]:=2;
                         4: pop[i,j]:=0;
                         6: pop[i,j]:=10;
                         8: pop[i,j]:=8;
                         10: pop[i,j]:=6;
                    end;
               end
               else
               if (mirr = 4) or (mirr = 10) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=8;
                         2: pop[i,j]:=6;
                         4: pop[i,j]:=4;
                         6: pop[i,j]:=2;
                         8: pop[i,j]:=0;
                         10: pop[i,j]:=10;
                    end;
               end;
          end;
     end;
end;
```

```
{------ Mirror between two random notes in all the population,taking----
 ------------------ the first note as a base----------------------------}
procedure mirror_first_part (var pop:population);
var
   i,j,mirr,pos1,pos2:integer;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          pos1:=0;
          while pos1 = 0 do
          begin
               pos1:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos1+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          mirr:=pop[i,1];
          for j:=pos1 to pos2 do
          begin
               if (mirr <> 12) and (mirr <> 14) then
               begin
                    if (mirr = 0) or (mirr = 6) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=0;
                              2: pop[i,j]:=10;
                              4: pop[i,j]:=8;
                              6: pop[i,j]:=6;
                              8: pop[i,j]:=4;
                              10: pop[i,j]:=2;
                         end;
                    end;
                    if (mirr = 2) or (mirr = 8) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=4;
                              2: pop[i,j]:=2;
                              4: pop[i,j]:=0;
                              6: pop[i,j]:=10;
                              8: pop[i,j]:=8;
                              10: pop[i,j]:=6;
                         end;
                    end;
                    if (mirr = 4) or (mirr = 10) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=8;
                              2: pop[i,j]:=6;
                              4: pop[i,j]:=4;
                              6: pop[i,j]:=2;
                              8: pop[i,j]:=0;
                              10: pop[i,j]:=10;
                         end;
                    end;
               end;
          end;
     end;
 end;
```

```
{----Mirror taking a random note like base between two random notes----
----------------in every melody of the population-------------}
procedure mirror_random_part (var pop:population);
var
   i,j,mirr,pos1,pos2:integer;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          pos1:=0;
          while pos1 = 0 do
          begin
               pos1:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos1+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          mirr:=random(11);
          while mirr mod 2 <> 0 do
          begin
               mirr:= random(11);
          end;
          for j:=pos1 to pos2 do
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=0;
                         2: pop[i,j]:=10;
                         4: pop[i,j]:=8;
                         6: pop[i,j]:=6;
                         8: pop[i,j]:=4;
                         10: pop[i,j]:=2;
                    end;
               end
               else
               if (mirr = 2) or (mirr = 8) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=4;
                         2: pop[i,j]:=2;
                         4: pop[i,j]:=0;
                         6: pop[i,j]:=10;
                         8: pop[i,j]:=8;
                         10: pop[i,j]:=6;
                    end;
               end
               else
               if (mirr = 4) or (mirr = 10) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=8;
                         2: pop[i,j]:=6;
                         4: pop[i,j]:=4;
                         6: pop[i,j]:=2;
                         8: pop[i,j]:=0;
                         10: pop[i,j]:=10;
                    end;
               end;
          end;
     end;
```

```
end;

{--------- Mirror + reverse in every melody of the population -------}
procedure Mirror_reverse_whole(var pop:population);
var
   i,j,k,mirr:integer;
   aux:melodies;
begin
     for i:=1 to maxpop do
     begin
          mirr:=pop[i,1];
          for j:=1 to maxstr do
          begin
               if (mirr <> 12) and (mirr <> 14) then
               begin
                    if (mirr = 0) or (mirr = 6) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=0;
                              2: pop[i,j]:=10;
                              4: pop[i,j]:=8;
                              6: pop[i,j]:=6;
                              8: pop[i,j]:=4;
                              10: pop[i,j]:=2;
                         end;
                    end;
                    if (mirr = 2) or (mirr = 8) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=4;
                              2: pop[i,j]:=2;
                              4: pop[i,j]:=0;
                              6: pop[i,j]:=10;
                              8: pop[i,j]:=8;
                              10: pop[i,j]:=6;
                         end;
                    end;
                    if (mirr = 4) or (mirr = 10) then
                    begin
                         case pop[i,j] of
                              0: pop[i,j]:=8;
                              2: pop[i,j]:=6;
                              4: pop[i,j]:=4;
                              6: pop[i,j]:=2;
                              8: pop[i,j]:=0;
                              10: pop[i,j]:=10;
                         end;
                    end;
               end;
               k:=j;
               aux[maxstr-k+1]:=pop[i,j];
          end;
          pop[i]:=aux;
     end;
end;

{Mirror + reverse between two random notes in every melodie of the population}
procedure Mirror_reverse_part(var pop:population);
var
   i,j,k,mirr,pos1,pos2,l:integer;
   aux:melodies;
begin
     randomize;
```

```
for i:=1 to maxpop do
begin
     pos1:=0;
     while pos1 = 0 do
     begin
          pos1:=random(maxstr-1);
     end;
     pos2:=0;
     while (pos2=0) or (pos2<=pos1+1) do
     begin
          pos2:=random(maxstr+1);
     end;
     mirr:=pop[i,1];
     for j:=pos1 to pos2 do
     begin
          if (mirr <> 12) and (mirr <> 14) then
          begin
               if (mirr = 0) or (mirr = 6) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=0;
                         2: pop[i,j]:=10;
                         4: pop[i,j]:=8;
                         6: pop[i,j]:=6;
                         8: pop[i,j]:=4;
                         10: pop[i,j]:=2;
                    end;
               end;
               if (mirr = 2) or (mirr = 8) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=4;
                         2: pop[i,j]:=2;
                         4: pop[i,j]:=0;
                         6: pop[i,j]:=10;
                         8: pop[i,j]:=8;
                         10: pop[i,j]:=6;
                    end;
               end;
               if (mirr = 4) or (mirr = 10) then
               begin
                    case pop[i,j] of
                         0: pop[i,j]:=8;
                         2: pop[i,j]:=6;
                         4: pop[i,j]:=4;
                         6: pop[i,j]:=2;
                         8: pop[i,j]:=0;
                         10: pop[i,j]:=10;
                    end;
               end;
          end;
     end;
     k:=1;
     for j:=pos2 downto pos1 do
     begin
          aux[k]:=pop[i,j];
          k:=k+1;
     end;
     k:=1;
     for l:=pos1 to pos2 do
     begin
          pop[i,l]:=aux[k];
          k:=k+1;
          end;
     end;
end;

{-----------Reverse in every melody of the population---------}
procedure reverse(var pop:population);
var
   i,j,k,l:integer;
   aux:melodies;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          k:=1;
          for j:=maxpop downto 1 do
          begin
               aux[k]:=pop[i,j];
               k:=k+1;
          end;
          k:=1;
          for l:=1 to maxstr do
          begin
               pop[i,l]:=aux[k];
               k:=k+1;
          end;
     end;
end;

{---------Reverse in a part of every melody of the population -----}
procedure reverse_part(var pop:population);
var
   pos1,pos2,i,j,k,l:integer;
   aux:melodies;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          pos1:=0;
          while pos1 = 0 do
          begin
               pos1:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos1+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          k:=1;
          for j:=pos2 downto pos1 do
          begin
               aux[k]:=pop[i,j];
               k:=k+1;
          end;
          k:=1;
          for l:=pos1 to pos2 do
          begin
               pop[i,l]:=aux[k];
               k:=k+1;
          end;
     end;
end;

{--------- transpose a random value  7 randomly selected melodies ------}
```

```
procedure transpose(var pop:population);
var
   i,j,k,trans,melod:integer;
   rep:new;
begin
     randomize;
     rep[1]:=120;
     for i:=1 to maxmelod do
     begin
          melod:=0;
          while melod=0 do
          begin
               melod:=random(maxstr+1);
          end;
          for k:=1 to i do
          begin
               if melod =rep[k] then
               begin
                    melod:=random(maxstr+1);
               end;
          end;
          rep[i]:=melod;
          trans:=random(7);
          trans:=trans*2;
          for j:=1 to maxstr do
          begin
               if pop[melod,j] <= 10 then
               begin
                    pop[melod,j]:=pop[melod,j]+trans;
                    if pop[melod,j] > 10 then
                    begin
                         pop[melod,j]:=pop[melod,j]-12;
                    end;
               end;
          end;
     end;
end;

{--------- Transpose a part of every melody a random value----------}
procedure transpose_part(var pop:population);
var
   i,j,k,pos1,pos2,trans:integer;
begin
     randomize;
     for i:=1 to maxpop do
     begin
          pos1:=0;
          while pos1 = 0 do
          begin
               pos1:=random(maxstr-1);
          end;
          pos2:=0;
          while (pos2=0) or (pos2<=pos1+1) do
          begin
               pos2:=random(maxstr+1);
          end;
          for j:=pos1 to pos2 do
          begin
               if pop[i,j] <= 10 then
               begin
                    pop[i,j]:=pop[i,j]+trans;
                    if pop[i,j] > 10 then
                    begin
```

```
                        pop[i,j]:=pop[i,j]-12;
                   end;
              end;
         end;
    end;
end;

{---------------swap two adjacent notes in every melody-----------}
procedure swap_two(var pop:population);
var
   i,j,pos,aux:integer;
begin
      randomize;
     for i:=1 to maxpop do
     begin
          pos:=0;
          while (pos=0) or (pop[i,pos]=14) do
          begin
               pos:=random(maxstr);
          end;
          aux:=pop[i,pos];
          if pop[i,pos+1] <>14 then
          begin
               pop[i,pos]:=pop[i,pos+1];
               pop[i,pos+1]:=aux;
          end
          else
          begin
               if pop[i,pos+2] <> 14 then
               begin
                    pop[i,pos]:=pop[i,pos+2];
                    pop[i,pos+2]:=aux;
               end
               else
                    pop[i,pos]:=pop[i,pos+3];
                    pop[i,pos+3]:=aux;
          end;
     end;
end;

{ ---swap all the adjacent notes of every melodie(1-2,3-4,5-6...-----}
procedure swap_all(var pop:population);
var
   i,j,aux:integer;
begin
     for i:=1 to maxpop do
     begin
          j:=1;
          while j < maxstr do
          begin
               aux:=pop[i,j];
               pop[i,j]:=pop[i,j+1];
               pop[i,j+1]:=aux;
               j:=j+2;
          end;
     end;
end;

{------------ swap all the notes on position to the left --------------}
procedure swap_left(var pop: population);
var
   i,j,aux:integer;
begin
```

```
     for i:=1 to maxpop do
     begin
          aux:=pop[i,1];
          for j:=1 to maxstr-1 do
          begin
               pop[i,j]:=pop[i,j+1];
          end;
          pop[i,maxstr]:=aux;
     end;
end;

{----------------------Shows the melody founded----------------------}
procedure display_best(var position:integer;
                       var ff_function: fitness);
var
   i:integer;
begin
     for i:=1 to maxstr do
     begin
          write(ff_function[position,i]);
          if (i=20) or (i=40) or (i=60) then
          begin
               writeln
          end
          else
              write(',')
     end;
end;

{---------Count procedure, and show if the melody is founded-----}
Procedure count (var gen:integer;
                var pun: allmel_pos;
                var pop:population);
var
   k: integer;
begin
     k:=1;
     while k <= maxpop do
     begin
          if punt[k,2]= maxstr then
          begin
               cont:=cont+1;
               if cont =1 then
               begin
                    clrscr;
                    writeln;
                    write(' The melody ');
                    write(punt[k,1]);
                    write(' have been found in ');
                    write(gen);
                    writeln(' generations.');
                    writeln;
                    writeln(' This is the melody :');
                    writeln;
                    display_best(k,f_function);
                    k:=81;
                    read(wait);
                    clrscr;
               end;
          end;
          k:=k+1;
     end;
end;
```

```
{---------------------------Menu-------------------------------------}
procedure menu(var generet:integer);
begin

     writeln('
**********************************************************');
     writeln('        *
*');
     writeln('        *                    Genetic Algorithm
*');
     writeln('        *
*');
     writeln('        *                              Damian Padr¢n Abrante - 2005
*');
     writeln('        *
*');
     writeln('
**********************************************************');
     writeln;
     writeln('  Press a key to continue...');
     read(wait);
     clrscr;
     writeln('        ************************* MENU
***************************');
     writeln('        *
*');
     writeln('        * - Number of melodies/chromosomes: 100 (by default)
*');
     writeln('        *   (can be modified in the code between 10-180).
*');
     writeln('        *
*');
     writeln('        * - Number of bars: 10 (4/4)
*');
     writeln('        *
*');
     writeln('        * - Number of melodies of the fitness function: 7
*');
     writeln('        *
*');
     writeln('        * - Minimum duration of a note: quaver.
*');
     writeln('        *
*');
     writeln('        * - Maximum duration of a note: minim.
*');
     writeln('        *
*');
     writeln('        * - Lowest note: C#4 (midi-number-key = 61).
*');
     writeln('        *
*');
     writeln('        * - Highest note: A4 (midi-number-key = 69).
*');
     writeln('        *
*');
     writeln('
**********************************************************');
     writeln;
     writeln(' Introduce the number of generations (10-1000): ');
     readln(gener);
end;
```

```
{--------------------Menu 1 - Class 1-------------------------}
procedure Menu1(var opt: char);

begin
     writeln(' Choose an option : ');
     writeln;
     writeln(' a) Mutation ');
     writeln(' b) One point crossover and mutation ');
     writeln(' c) Two point crossover and mutation ');
     readln(opt);
end;

{------------------Menu2 - Class 2----------------}
procedure Menu2(var opt: char);

begin
     writeln(' Choose an option : ');
     writeln;
     writeln(' d) One point crossover with transpose and classic mutation ');
     writeln(' e) One point crossover with transpose and rhythm mutation');
     writeln(' f) One point crossover with reverse and classic mutation ');
     writeln(' g) One point crossover with reverse and rhythm mutation ');
     writeln(' h) One point crossover with mirror and classic mutation ');
     writeln(' i) One point crossover with mirror and rhythm mutation');
     writeln;
     writeln(' j) Two point crossover with transpose and classic mutation ');
     writeln(' k) Two point crossover with transpose and rhythm mutation');
     writeln(' l) Two point crossover with reverse and classic mutation ');
     writeln(' m) Two point crossover with reverse and rhythm mutation ');
     writeln(' n) Two point crossover with mirror and classic mutation ');
     writeln(' o) Two point crossover with mirror and rhythm mutation');
     writeln;
     writeln(' p) One point crossover with mirror, reverse  and classic mutation
');
     writeln(' q) Two point crossover with mirror, reverse  and classic
mutation');
     writeln(' r) One point crossover with mirror, reverse, transpose and classic
mutation ');
     writeln(' s) Two point crossover with mirror, reverse, transpose and classic
mutation ');
     writeln(' t) One point crossover with mirror, reverse  and rhythm mutation
');
     writeln(' u) Two point crossover with mirror, reverse  and rhythm
mutation');
     writeln(' v) One point crossover with mirror, reverse, transpose and rhythm
mutation ');
     writeln(' w) Two point crossover with mirror, reverse, transpose and rhythm
mutation ');
     readln(opt);
end;

{------------------Menu3 - Class 3---------------}
procedure Menu3(var opt: char);
begin
     writeln('Choose an option : ');
     writeln;
     writeln(' NOTE: There are three possibilities for each musical function:');
     writeln;
     writeln(' First column character: The specified function.');
     writeln(' Second column character: Specified function adding the classic
mutation(CMut).');
     writeln(' Third column character: Specified function adding the rhythm
mutation(RMut).');
```

```
     writeln;
     writeln('   CMut RMut');
     writeln(' a - n - 0) Mirror in each melody of the population (random mirror
axis)');
     writeln(' b - o - 1) Mirror in each melody of the population (first note as
mirror axis)');
     writeln(' c - p - 2) Mirror in a random part of each melody (random mirror
axis)');
     writeln(' d - q - 3) Mirror in a random part of each melody (first note as
mirror axis) ');
     writeln(' e - r - 4) Reverse in each melody of the population ');
     writeln(' f - s - 5) Reverse in a random part of each melody of the
population ');
     writeln(' g - t - 6) Mirror + reverse in each melody (mirror axis = first
note)');
     writeln(' h - u - 7) Mirror + reverse in a part of each melody (mirror
axis=first note)');
     writeln(' i - v - 8) Swap the pitch of two adjacent notes ');
     writeln(' j - w - 9) Swap all the adjacent integers of each melody of the
population ');
     writeln(' k - x - $) Swap all the integers one place leftwards ');
     writeln(' l - y - &) Transpose seven melodies randomly selected of the
population ');
     writeln(' m - z - #) Transpose a part of each melody of the population ');
     readln(opt3);
end;

{------------------------- Main program ---------------------------}
begin
     1: cont:=0;
     while opt2 <> 'a' do
     begin
          clrscr;
          menu(gener);
          fill_fitness(f_function);
          Generate_pop(pop1);
          writeln;
          clrscr;
          writeln(' Introduce the kind of operators/functions that you want to
use: ');
          writeln;
          writeln(' Class 1 : Classic operators (One/two point crossover -
Mutation)');
          writeln(' Class 2 : Classic operators mixed with musical functions');
          writeln(' Class 3 : Musical functions');
          readln(class);
          clrscr;
          opt1:='?';
          case class of
               '1': menu1(opt1);
               '2': menu2(opt1);
               '3': menu3(opt3);
          end;
          writeln;
          clrscr;
          write(' Processing. ');
          for i:=1 to gener do
          begin
               proc:=i div 100;
               if (proc mod 2= 0) then
               begin
                    write('. ');
               end;
```

```
write;
note_to_note(pop1,f_function,value);
save_pos_mel (value,punt);
bestt(punt,pop1,pop2);
if i=gener then
begin
     fin:=punt;
end;
copy1(pop1,pop2);
if opt1 <> '?' then
begin
case opt1 of
     'a': mutation(pop1);
     'b': begin
               one_crossover(pop1);
               mutation(pop1);
          end;
     'c': begin
               two_crossover(pop1);
               mutation(pop1);
          end;
     'd': begin
               one_crossover_transp(pop1);
               mutation(pop1);
          end;
     'e': begin
               one_crossover_transp(pop1);
               mutation_rhythm1(pop1);
               mutation_rhythm2(pop1);
          end;
     'f': begin
               one_crossover_rever(pop1);
               mutation(pop1);
          end;
     'g': begin
               one_crossover_rever(pop1);
               mutation_rhythm1(pop1);
               mutation_rhythm2(pop1);
          end;
     'h': begin
               one_crossover_mirr(pop1);
               mutation(pop1);
          end;
     'i': begin
               one_crossover_mirr(pop1);
               mutation_rhythm1(pop1);
               mutation_rhythm2(pop1);
          end;
     'j': begin
               Two_crossover_transp(pop1);
               mutation(pop1);
          end;
     'k': begin
               Two_crossover_transp(pop1);
               mutation_rhythm1(pop1);
               mutation_rhythm2(pop1);
          end;
     'l': begin
               Two_crossover_rever(pop1);
               mutation(pop1);
          end;
     'm': begin
               Two_crossover_rever(pop1);
```

```
                        mutation_rhythm1(pop1);
                        mutation_rhythm2(pop1);
                 end;
          'n': begin
                        Two_crossover_mirr(pop1);
                        mutation(pop1);
                 end;
          'o': begin
                        Two_crossover_mirr(pop1);
                        mutation_rhythm1(pop1);
                        mutation_rhythm2(pop1);
                 end;
          'p': begin
                        one_crossover_mirr_rever(pop1);
                        mutation(pop1);
                 end;
          'q': begin
                        two_crossover_mirr_rever(pop1);
                        Mutation(pop1);
                 end;
          'r': begin
                        one_crossover_mirr_rever_transp(pop1);
                        Mutation(pop1);
                 end;
          's': begin
                        two_crossover_mirr_rever_transp(pop1);
                        Mutation(pop1);
                 end;
          't': begin
                        one_crossover_mirr_rever(pop1);
                        Mutation_rhythm1(pop1);
                        mutation_rhythm2(pop1);
                 end;
          'u': begin
                        two_crossover_mirr_rever(pop1);
                        Mutation_rhythm1(pop1);
                        mutation_rhythm2(pop1);
                 end;
          'v': begin
                        one_crossover_mirr_rever_transp(pop1);
                        Mutation_rhythm1(pop1);
                        Mutation(pop1);
                 end;
          'w': begin
                        two_crossover_mirr_rever_transp(pop1);
                        Mutation_rhythm1(pop1);
                        mutation_rhythm2(pop1);
                 end

          end;
     end
     else
     case opt3 of
         'a': mirror_random(pop1);
         'b': mirror_first(pop1);
         'c': mirror_random_part(pop1);
         'd': mirror_first_part(pop1);
         'e': reverse(pop1);
         'f': reverse_part(pop1);
         'g': Mirror_reverse_whole(pop1);
         'h': Mirror_reverse_part(pop1);
         'i': swap_two(pop1);
         'j': swap_all(pop1);
```

```
'k': swap_left(pop1);
'l': transpose(pop1);
'm': transpose_part(pop1);
'n': begin
           mutation(pop1);
           mirror_random(pop1);

     end;
'o': begin
           mutation(pop1);
           mirror_first(pop1);
     end;
'p': begin
           mirror_random_part(pop1);
           mutation(pop1);
     end;
'q': begin
           mirror_first_part(pop1);
           mutation(pop1);
     end;
'r': begin
           reverse(pop1);
           mutation(pop1);
     end;
's': begin
           reverse_part(pop1);
           mutation(pop1);
     end;
't': begin
           Mirror_reverse_whole(pop1);
           mutation(pop1);
     end;
'u': begin
           Mirror_reverse_part(pop1);
           mutation(pop1);
     end;
'v': begin
           swap_two(pop1);
           mutation(pop1);
     end;
'w': begin
           swap_all(pop1);
           mutation(pop1);
     end;
'x': begin
           swap_left(pop1);
           mutation(pop1);
     end;
'y': begin
           transpose(pop1);
           mutation(pop1);
     end;
'z': begin
           transpose_part(pop1);
           mutation(pop1);
     end;
'0': begin
           mirror_random(pop1);
           Mutation_rhythm1(pop1);
           Mutation_rhythm2(pop1);
     end;
'1': begin
           mirror_first(pop1);
```

```
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '2': begin
                    mirror_random_part(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '3': begin
                    mirror_first_part(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '4': begin
                    reverse(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '5': begin
                    reverse_part(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '6': begin
                    Mirror_reverse_whole(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '7': begin
                    Mirror_reverse_part(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '8': begin
                    swap_two(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '9': begin
                    swap_all(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '$': begin
                    swap_left(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '&': begin
                    transpose(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
          '#': begin
                    transpose_part(pop1);
                    Mutation_rhythm1(pop1);
                    Mutation_rhythm2(pop1);
               end;
end;
generate2(pop1);
note_to_note(pop1,f_function,value);
save_pos_mel (value,punt);
bestt2(punt,pop1,pop2);
```

```
                copy2(pop1,pop2);
                count (i,punt,pop1);
                if cont > 1 then
                begin
                     writeln;
                     write(' Besides,in this generation, ');
                     write(cont);
                     writeln(' identical melodies have been found.');
                     writeln;
                     writeln(' Press a key to continue... ');
                     readln(wait);
                     goto 2;
                end;
           end;
           clrscr;
           writeln(' No identical melody has been found... ');
           writeln;
           write(' But the most similar melody has a fitness value of: ');
           writeln(fin[1,2]);
           writeln;
           write(' With regard to the melody: ');
           writeln(fin[1,1]);
           writeln;
           writeln(' Press a key to continue... ');
           read(wait);
           2: clrscr;
           opt2:='z';
           while (opt2 <> 'a') and (opt2 <> 'b') do
           begin
                writeln(' What do you want to do? ');
                writeln;
                writeln(' a) Exit ');
                writeln(' b) Restart the program ');
                writeln;
                writeln(' Choose an option: ');
                read(opt2);
                if (opt2 <> 'a') and (opt2 <> 'b') then
                begin
                     clrscr;
                end;
           end;
           if opt2 = 'b' then
           begin
                goto 1;
           end;
      end;
end.
```

Bibliography:

[Ariza 2002] Ariza, Christopher: "Prokaryotic Groove Rhythmic Cycles as Real-Value Encoded Genetic Algorithms." In: Proceedings of the International Computer Music Conference. San Francisco: International Computer Music Association. 2002. pp. 561-567.

[Banzhaf 1998] Banzhaf, Wolfgang; Nordin , Peter; Keller, Robert E; Francone, Frank D: "Genetic Programming, An Introduction: On the Automatic Evolution of Computer Programs and Its Applications." Morgan Kaufmann Publishers, Inc. 1998.

[Bentley 1997] Bentley, Peter: "The Revolution of Evolution for Real-World Applications". In: Emerging Technologies '97: Theory and application of Evolutionary Computation. University College London, UK. 1997. pp. 1-11.

[Biles 1994] Biles, J. A.: "GenJam: A genetic algorithm for generating jazz solos". In: Proceedings of the 1994 International Computer Music Conference. The International Computer Music Association, San Francisco, CA. 1994. pp. 131-137.

[Biles 1995] Biles, J. A.: "GenJam Populi: Training an IGA via audience-mediated Performance." In: Proceedings of the 1995 International Computer Music Conference. The International Computer Music Association, San Francisco, CA. 1995. pp. 347-348

[Biles 1996] Biles, J.A.; Anderson, P.G.; Loggi, L.W.: "Neural network fitness functions for a musical GA". In: Proceedings of the International ICSC Symposium on Intelligent Industrial Automation (IIA'96) and Soft Computing (SOCO'96). Academic Press, Reading, UK: ICSC.1996. pp. B39-B44.

[Biles 1998] Biles, J. A.: "Interactive GenJam: Integrating Real-Time Performance with a Genetic Algorithm". In: Proceedings of the 1998 International Computer Music Conference. The International Computer Music Association, San Francisco, CA.1998. pp. 232-235.

[Burton 1997] Burton A., Vladimirova, T.: "Applications of Genetic Techniques to Musical Composition". Unpublished manuscript. Last Update: 1997, Last visit: 24.06.05, URL: www.tony-b.freeuk.com/docs/cmjcompo_doc.uue,

[Burton 1998] Burton, A. R.: "A Hybrid Neuro-Genetic Pattern Evolution System Applied to Musical Composition", Ph.D. dissertation. University of Surrey. 1998.

[Collins 2000] Collins, Trevor D.: "The Application of Software Visualization Technology to Evolutionary Computation: A Case Study in Genetic Algorithms". Thesis. Chapter 2: An Overview of Evolutionary Computation. Last Update: 2000, Last Visit: 31.03.05, URL: http://kmi.open.ac.uk/people/trevor/archive/thesis/

[Darwin 1859] Darwin, C.: "On the origins of species by means of natural selection or the preservation of favoured races in the struggle for life". Murray, London, UK. 1859

[De Jong 1975] De Jong, K.A.: "An analysis of the behavior of a class of genetic adaptive systems." Ph.D. dissertation, University of Michigan, Ann Arbor.1975

[Fogel 1966] Fogel, L.; Owens, A; Wanls, M.: "Artificial Intelligence through Simulated Evolution". Wiley, New York. 1966.

[Gartland-Jones 2002] Gartland-Jones, A.: "Can a Genetic Algorithm Think Like a Composer". In: Proceedings of 5th International Conference on Generative Art, Politecnico di Milano University, Milan. 2002. pp.14.1-14.12.

[Gartland-Jones 2003a] Gartland-Jones, A.: "MusicBlox: a Real-time Algorithmic Composition System Incorporating a Distributed Interactive Genetic Algorithm". In: Proceedings of EvoWorkshops/EuroGP2003, 6th European Conference on Genetic Programming. 2003. pp. 490-501.

[Gartland-Jones 2003b] Gartland-Jones, A.; Copley, Peter: "The Suitability of Genetic Algorithms for Musical Composition". In: Contemporary Music Review, VOL. 22, No. 3, 2003. pp. 43–55.

[Gibson 1991] Gibson, P. M.; Byrne, J. M.: "Neurogen: Musical composition using genetic algorithms and cooperating neural networks". In: Proceedings of the Second International Conference on Artificial Neural Networks, London.1991. IEE. pp. 309–313

[Goldberg 1989] Goldberg, David E.: "Genetic Algorithms in Search, Optimization, and Machine Learning". Addison Wesley, 1989

[González 2003] González, Fabio, Ph.D.; Hernández, Germán Ph.D.; Camargo, Carlos M.Sc.: "Introducción a la Computación Evolutiva". Last update: 2003, Last Visit: 03.04.2005, URL: http://.dis.unal.edu.co/~fgonza/courses/2003/pmge/introEvol.pdf

[Holland 1975] Holland, John: "Adaptation in natural and artificial systems". University of Michigan Press, 1975.

[Horner 1991] Horner, A.; Goldberg, D. E.: "Genetic algorithms and computer-assisted music composition". In: Proceedings of the International Computer Music Conference, 1991. The International Computer Music Association, San Francisco, CA. 1991. pp. 479-482.

[Horowitz 1994] Horowitz, D.: "Generating Rhythms with Genetic Algorithms". In: Proceedings of the International Computer Music Conference. The International computer Music Association. San Francisco, CA. 1994. pp. 142-143.

[Jacob 1994] Jacob, Bruce L.: "Composing with Genetic Algorithms". In: Proceedings of the International Computer Music Conference. 1994. pp. 452-455.

[Jacob 1996] Jacob, Bruce L.: "Algorithmic Composition as a Model of Creativity". In: Organised Sound, vol. 1, no 3. December, 1996. pp. 157-165.

[Johanson 1998] Johanson, B.; Poli, R.: "Gp-music: An interactive genetic programming system for music generation with automated fitness raters". In: Proceedings of the 3rd International Conference on Genetic Programming, GP'98. University of Wisconsin, Madison, Wisconsin, USA. 1998. pp.181–186.

[Johnson 1999] Johnson, C.: "Exploring the Sound-Space of Synthesis Algorithms Using Interactive Genetic Algorithms". In: AISB'99 Symposium on Musical Creativity. Society for Artificial Intelligence and the Simulation of Behaviour. Edinburgh. 1999. pp. 20-27.

[Koza 1992] Koza, John: "Genetic Programming: On the Programming of Computers by Means of Natural Selection". The MIT Press, Cambridge, MA. 1992.

[Mc Auley 2003] McAuley Tristan; Hingston, Philip: "Algorithmic Composition in Contrasting Music Styles". In: Proceedings of the ICMC 2003, International Computer Music Conference. September 2003.

[Mc Intyre 1994] Mc Intyre, R. A.: "Bach in a Box: The Evolution of Four-Part Baroque Harmony Using Genetic Algorithm". In: Proceedings of the IEEE Conference on Evolutionary Computation. IEEE Press. Washington DC. 1994. pp. 852-857.

[Miranda 1995] Miranda, E.: "Granular Synthesis of Sounds by Means of a Cellular Automaton". In: Leonardo, Vol. 28 (4). The MIT Press, Cambridge, MA.1995.

[Moroni 2000] Moroni, A.; Manzolli, J.; Von Zuben, Fernando: "Composing with interactive genetic algorithms". In: Proceedings of the SBC2000 - Congresso da Sociedade Brasileira de Computaçao. 2000.

[Moroni 2000b] Moroni, A.; Manzolli, J. ; Von Zuben, F. ; Gudwin, R.: “Vox Populi: An Interactive Evolutionary System for Algorithmic Music Composition”, In: Leonardo Music Journal - MIT Press, Vol. 10.San Francisco, USA. 2000. pp. 49–54.

[Papadopoulos 1998] Papadopoulos, George; Wiggins, Geraint: “A Genetic Algorithm for the Generation of Jazz Melodies”. In: STeP′98, Jyväskylä, Finland. 1998.

[Papadopoulos 1999] Papadopoulos, G.; Wiggins, G.: “AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects”. In: Proceeding of: AISB’99 Symposium on Musical Creativity. Society for Artificial Intelligence and the Simulation of Behaviour, Edinburgh. 1999. pp. 110-117.

[Pazos 1999a] Pazos, A., Santos, A., Dorado, J., Romero, J. J.: “Adaptive Aspects of Rhythmic Composition: Genetic Music”. Banzhaf, W; Daida, J; Eiben, A.E; Garzon, M.H; Honavar, V; Jakiela, M; Smith, R.E. (Eds.). In: Proceedings of the Genetic and Evolutionary Computation Conference GECCO'99. Vol. 2 (pp. 1794). Morgan Kaufmann. San Francisco, CA. 1999.

[Pazos 1999b] Pazos, A., Santos, A., Dorado, J., Romero, J.: “Genetic Music Compositor”. Angeline, P; Michalewicz, Z; Schoenhauer, M; Yao X; Zalzala, A. (Eds.). In: Proceedings of the 1999 Congress on Evolutionary Computation Vol. 2. IEEE Press. Washington DC. 1999. pp. 885 - 890.

[Pazos 1999c] Pazos, Alejandro; Romero-Carralda, J.J.: “Musical Adaptive Systems”. In: Proceedings of the Student Workshop-Genetic and Evolutionary Computation Conference 1999 (GECCO'99). Morgan Kaufmann. San Francisco, CA. 1999. pp. 343- 344

[Pfeifer 2003] Pfeifer, Rolf; Fend, Miriam; Krafft Martin F.: “Artificial Life”. Thesis. Chapter 6: Artificial Evolution. Last update: 2004, Last Visit: 03.04.2005, URL: http://www.ifi.unizh.ch/ailab/teaching/AL04/lecture/chap6.pdf.

[Phon- Amnuaisuk 1999a] Phon-Amnuaisuk, S.; Tuson, Andrew; Wiggins, Geraint A.: "Evolving Musical Harmonisation". In: Proceedings of the International Conference on Adaptive and Natural Computing Algorithms, Porto Roz, Slovenia. 1999.

[Phon-Amnuaisuk 1999b] Phon-Amnuaisuk, S., Wiggins, G. A.: "The Four-Part Harmonisation Problem: A comparison between Genetic Algorithms and a Rule-Based System". In: Proceedings of the AISB'99 Symposium on Musical Creativity. Society for Artificial Intelligence and the Simulation of Behaviour. Edinburgh. 1999. pp. 28-34.

[Pigg 2002] Pigg, Paul: "Cohesive Music Generation with Genetic Algorithms." Last Update: 2002. Last Visit: 10.04.2005.URL: http://web.umr.edu/~tauritzd/courses/cs401/fs2002/project/Pigg.pdf

[Ralley 1995] Ralley, David: "Genetic algorithms as a tool for melodic development". In: Proceedings of the 1995 International Computer Music Conference, San Francisco. 1995. pp. 501-502.

[Rechenberg 1973] Rechenberg, Ingo.: "Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der biologischen Evolution". Stuttgart, Fromman-Holzboog Verlag. 1973.

[Rechenberg 1965] Rechenberg, Ingo.: "Cybernetic solution path of an experimental problem: Kybernetische Lösungsansteuerung einer experimentellen forschungsaufgabe". English translation of a German technical report RAE/LT-1122 (Report-C), Held by Cranfield University. 1965.

[Santos 2000] Santos, Antonino; Arcay, Bernardino; Dorado, Julián; Romero, Juan; Rodriguez, Jose: "Evolutionary Computation Systems for Musical Composition". In: Mathematics and Computers in Modern Science, Worlds Scientific and Engineering Society Press. 2000. pp. 97-102.

[Schwefel 1968] Schwefel, Hans Paul: "Experimentelle Optimierung einer Zweiphasendüse". Teil I. Technical Report No. 35 of the Project MHD-Staustrahlrohr 11.034/68, AEG Research Institute, Berlin. 1968

[Schwefel 1975] Schwefel, Hans Paul: "Evolutionsstrategie und numerische Optimierung". Dissertation. Technische Universitat Berlin, Berlin. 1975.

[Schwefel 1977] Schwefel, Hans Paul: "Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie". Basel, Birkenhäuser. 1977.

[Schwefel 1995] Schwefel, Hans Paul.: "Evolution and Optimum Seeking". New York: Wiley. 1995.

[Spector 1995] Spector, Lee: "Evolving Control Structures with Automatically Defined Macros". In: Working Notes of the AAAI Fall Symposium on Genetic Programming. The American Association for Artificial Intelligence. 1995.

[Spector 1995b] Spector, Lee; Alpern, Adam: "Induction and Recapitulation of Deep Musical Structure". In: Proceedings of the: Working Notes of the IJCAI-95 Workshop on Artificial Intelligence and Music. 1995. pp. 41-48.

[Sundberg 1976] Sundberg, Johan; Lindblom, Bjorn: "Generative Theories in Language and Music Descriptions". In: Schwanauer, Stephan M.; Levitt, David A.: Machine Models of Music. The MIT Press, Cambridge, Massachusetts. London, England. 1993. pp. 263-286.

[Takala 1993] Takala, Tapio; Hahn, James; Gritz, Larry; Geigel ,Joe; Won Lee, Jong: "Using Physically-Based Models and Genetic Algorithms for Functional Composition of Sound Signals, Synchronized to animated motion". In: Proceedings of the International Computer Music Conference. Tokyo, Japan. 1993. pp. 180-185.

[Todd 1999] Todd, Peter .M.; Werner, Gregory M.: "Frankensteinian methods for evolutionary music composition". In: Griffith, Niall; Todd, Peter .M. (Eds.): Musical networks: Parallel distributed perception and performance Cambridge: The MIT Press. 1999. pp. 313-339.

[Torres 2005] Torres, Juan Esteban; Martínez, José Jesús: "Tutorial de Programación Genética". Last update: 2005, Last Visit: 09.04.2005 URL: http://platon.escet.urjc.es/~ciic05/ciic2005/enlaces/documentos/Tutorial%20PG.pdf

[Towsey 2001] Towsey, Michael; Brown, Andrew; Wright, Susan; Diederich, Joachim: "Towards Melodic Extension Using Genetic Algorithms". In: Educational Technology & Society 4(2). 2001. pp. 54-65.

[Velikonja 2003] Velikonja, Peter: "Autonomous Music via Artificial Evolution". Thesis. Princeton University. 2003.

[Werner 1997] Werner, G.M. and Todd, P. M.: "Too many love songs: Sexual selection and the evolution of communication". In: Proceedings of the Forth European Conference on Artificial Life. The MIT Press, Cambridge, MA. 1997. pp. 434-443.

[Wiggins 1999] Wiggins, Geraint; Papadopoulos, George; Phon-Amnuaisuk, Somnuk; Tuson, Andrew: "Evolutionary Methods for Musical Composition". In: Proceedings of the International Journal of Computing Anticipatory Systems. 1999.