2019

AN INTELLIGENT DESIGN EXPLORER FOR NEW VIOLIN SHAPES

Hao Wang
University of Rhode Island, hao_wang@my.uri.edu

AN INTELLIGENT DESIGN EXPLORER FOR NEW VIOLIN SHAPES

BY

HAO WANG

MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF RHODE ISLAND

2019

MASTER OF SCIENCE THESIS

OF

HAO WANG

APPROVED:

Thesis Committee:

Major Professor	Lutz Hamel
	Jean-Yves Hervé
	Ben Anderson
	Nasser H. Zawia DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2019

ABSTRACT

The origins of evolutionary computation can be traced back to the late 1950s and started to receive significant attention during the 1980s. Pioneers have done a lot of research by using evolutionary computation, especially in the field of design, music, and art. This article focuses on the violin outline design, which is a very tiny segment of design. By using the evolution strategies algorithm, the violin explorer, a violin design tool has been developed. Meanwhile, by introducing user interaction steps, the violin explorer became a creative evolutionary system. With this system, ordinary people without any design knowledge can create their favorite violin outline by simple choices.

In addition, after reviewed by Institutional Review Board, an anonymous experiment was held to verify the usability of the violin explorer. Hundreds of participants with different ages and backgrounds were invited to use the violin explorer. During the anonymous experiment, the violin explorer generated a lot of creative violin outline designs, which are different from the original violin. Finally, the experimental data shows that there might be a connection between the age and background of the participants and their design.

ACKNOWLEDGMENTS

I would like to acknowledge many individuals for helping me during my master's study at the University of Rhode Island. Particularly, I would like to express my sincere gratitude to my advisor Dr. Lutz Hamel to provide the opportunity, knowledge, and support that enabled this study. I would also like to thank him for his continuous support, patience, and guidance throughout the past two years.

I would also like to thank my committee members Dr. Jean-Yves Hervé and Dr. Ben Anderson for generously offering their time, suggestions and goodwill throughout the preparation and review of this document.

A special thanks to my family – my wife Qingyu, and my newborn son Damon – for their love and support. I would also like to thank my parents and parents-in-law for their love and prayers.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In the age of technology, the interest in combined human intelligence with the power of computers is raising. With the blooming of artificial intelligence techniques, there are plenty of applications that are constantly being developed and showing a very impressive outcome. Art and design, which are highly relying on human experience, professional knowledge and aesthetics, have also become popular research directions. It is a very interesting topic to explore whether a computer can fulfill the designing tasks. Can computers think/work as a human designer? Furthermore, can computers create like a human designer? We might answer this question by developing an intelligent design explorer and seeing if there are any impressive designs created by the computer.

Design, by definition, is a plan for arranging elements in such a way as best to accomplish a particular purpose [1]. In a broader sense, a design is an applied art and engineering that integrates with technology [2]. It involves combinations of the visual arts disciplines, sciences, and technology, and requires problem-solving and communication skills [3]. A designer can be treated as one of the most important parts in such activity, which requires both professional knowledge and human creativity during the work. Their work is usually a high intersection of art, business, and technology knowledge. A good design usually takes into account many factors such as function, value, aesthetics and usage environments. That makes design a complex task. But with the progress of science and technology, the situation is changing.

With the development of computer technology, experts in computer science have developed a wide range of design software to free designers from their complex design work. This software usually plays the role of a facilitator or an assistant. By using this software, designers can make up for the lack of knowledge and reduce the complexity of their design work. Even though, with the help of this software, it is still considered "a mission impossible" for an ordinary person without any expert design knowledge to complete design work. The most difficult part is to make the computer "think" like a designer and offer advice to an ordinary person. Artificial intelligence may be a potential solution to this problem. Since the birth of artificial intelligence, theory and technology have become more and more mature, and the field of application has been expanding. Art and design are one of the popular sub-fields in artificial intelligence.

There are increasing numbers of researchers focusing on art and design areas, especially in areas such as architecture, engineering design, and aesthetic design. The algorithmic art assembly [4] hosted in San Francisco showcasing a diverse range of artists who are using algorithmic tools and processes in their works. Evolutionary computation, as a mature algorithm, also plays an important role in the art and design area. For example, John Frazer showed how evolution could generate many surprising and inspirational architectural forms and how novel and useful structures could be evolved [5]. Celestino Soddu uses evolution to generate everything from novel table lamps to castles to three-dimensional Picasso sculptures [6].

Evolutionary computation is a sub-domain of artificial intelligence. The original idea of evolutionary computation can be traced back to the end of the 1950s (Friedberg [7], [8], Box [9]). Evolutionary computation is influenced by the natural selection mechanism of the "survival of the fittest." The essence of evolutionary computation is to use evolutionary algorithms to search from a massive collection of potential solutions and find the global optimal solution [10]. By several decades of development and evolution, evolutionary algorithms developed into a family of algorithms.

Evolutionary algorithm, from a mathematical point of view, is essentially a search optimization method. Evolutionary algorithms have been widely used in many fields, such as pattern recognition, image processing, artificial intelligence, economic management, and so on. Furthermore, we can form a creative evolutionary system by using evolutionary algorithms.

A creative evolutionary system is a computer system that makes use of some aspect of evolutionary computation [10]. The difference between a creative evolutionary system and an evolutionary algorithm is: an evolutionary algorithm is used to search for an optimal solution; A creative evolutionary system acts as an explorer to provide inspiration and investigate creative solutions. The creative evolutionary system is been frequently used in the area such as architecture, engineering design, and aesthetic design. Compared to original evolutionary algorithms, we add human interaction to the evolutionary process. The judgment made by a human can contribute to our system and help us find a novel solution. Under this architecture, Our system can act "creatively." It might generate some highly innovate solutions or combine two very different ideas to create new ideas. Users can sidestep the limitation of "conventional wisdom" and "design fixation" by using this approach.

The purpose of this thesis is to construct a creative evolutionary system to work out a real-world problem. This study focuses on a very tiny segment of art and design: the violin outline design. The first goal is to develop a software that applies evolutionary algorithms to evolve and explore a new design of the violin. A human intersection process will be added to this software. Ordinary people can create their favorite violin by simply liking or disliking the violin designed by the computer. The second goal is to discuss and evaluate the creative ability of this software. The most difficult aspects of a design problem are people. Some factors such as fashion, age, background, etc. might affect people's design. We will invite 100 participants to anonymously use this software and see if there are any novel violin outlines designed. All violin drawing work in this study will be handled by Digital Amati software. Digital Amati [11], developed by Harry Mairson, is a software based on Euclidean geometry and uses "geometry engine" language (AmatiML) to design classical stringed instruments.

This is a brand new attempt in the field of art and design. After evolving and exploring new designs of the violin, users can also implement their design onto a Computer Numerical Control (CNC) machine to customize their own violin. Furthermore, these kinds of studies can be applied to design the geometric appearance of other items.

CHAPTER 2

REVIEW OF LITERATURE

2.1 Evolutionary Computation

Evolutionary Computation is a kind of effective bionic algorithm, which roots firmly in evolutionary biology and computer science (Figure 1) [10]. From the perspective of the computer field, evolutionary computation is all about search [12]. It defines a computational problem in terms of a search space, which can be viewed as a massive collection of potential solutions to the problem, and a point in that space defines a solution [12].

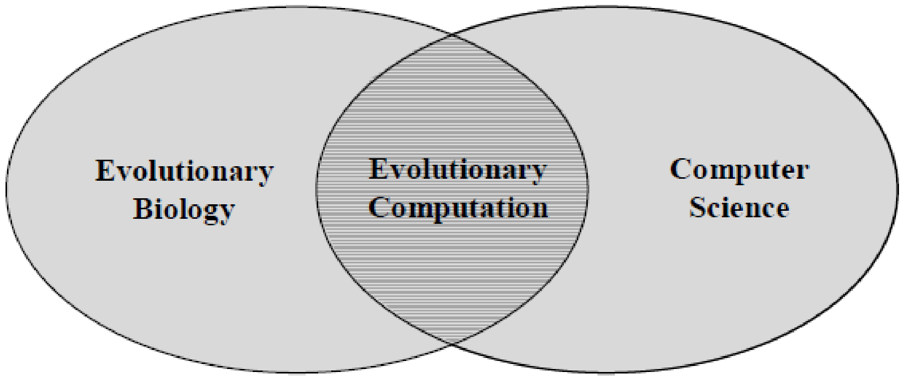


Figure 1. Evolutionary computation has its roots in computer science and evolutionary biology

On the other hand, it has a very close relationship with biological evolution. It is a study of computational systems that use ideas and get inspiration from natural evolution and adaptation [13]. It is typically used as an analogy with natural evolution to perform searches by evolving solutions to problems. And instead of working with one solution at a time in the search space, these algorithms consider a large collection or population of solutions at once [14].

The basic idea of evolutionary computation begins with a group of randomly generated initial populations, by following the inheritance of creatures, using reproduction, recombination and mutation methods to generate the next generation of populations. Then eliminate those individuals with a lower fitness to improve the quality of the new generation of the population. After repeated iterations, the optimal solution is gradually approached.

2.2 Evolutionary Algorithm

The field of evolutionary algorithms has grown up around four main members: genetic algorithms (GA), created by John Holland [15]; evolutionary programming (EP), created by Lawrence Fogel and developed further by his son David Fogel [16]; evolution strategies (ES), created by Ingo Rechenberg [17] and genetic programming (GP) created by John Koza [18].

The genetic algorithm is perhaps the most well known of all evolution-based search algorithms [14]. It maintains a population of individuals where each individual consists of a genotype (search space) and a corresponding phenotype (solution space). These two spaces have a one-to-one correspondence relationship (Figure 2).

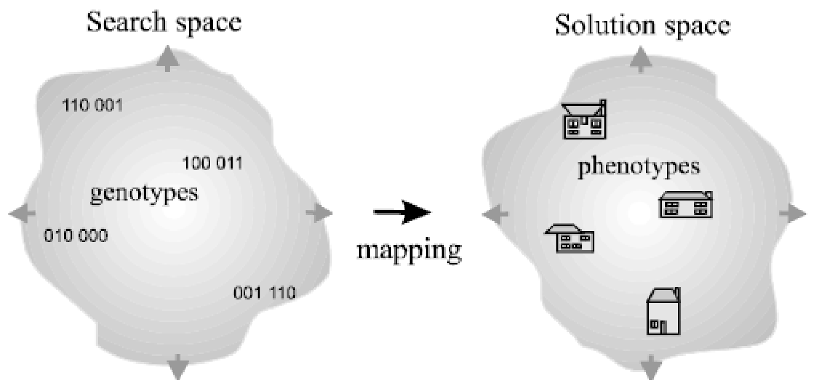


Figure 2. Mapping genotypes in the search space to phenotypes in the solution space

This algorithm works as below (Figure 3) [14]:

Step 1: The genotype of every individual in the population is initialized with random alleles.

Step 2: The main loop of the algorithm then begins, with the corresponding phenotype of every individual in the population being evaluated and given a fitness value according to how well it fulfills the problem objective or fitness function.

Step 3: Reproduce individuals according to their fitness, the higher fitness the more copies that are made of an individual. Then put these new individuals into a 'mating pool'.

Step 4: Randomly take two parents from the 'mating pool' and use a random crossover and mutate to generate two offspring. These processes will finish until enough new individuals fill the child population.

Step 5: Evolution terminates after a predefined number of generations, or when a solution of sufficient quality has been generated.

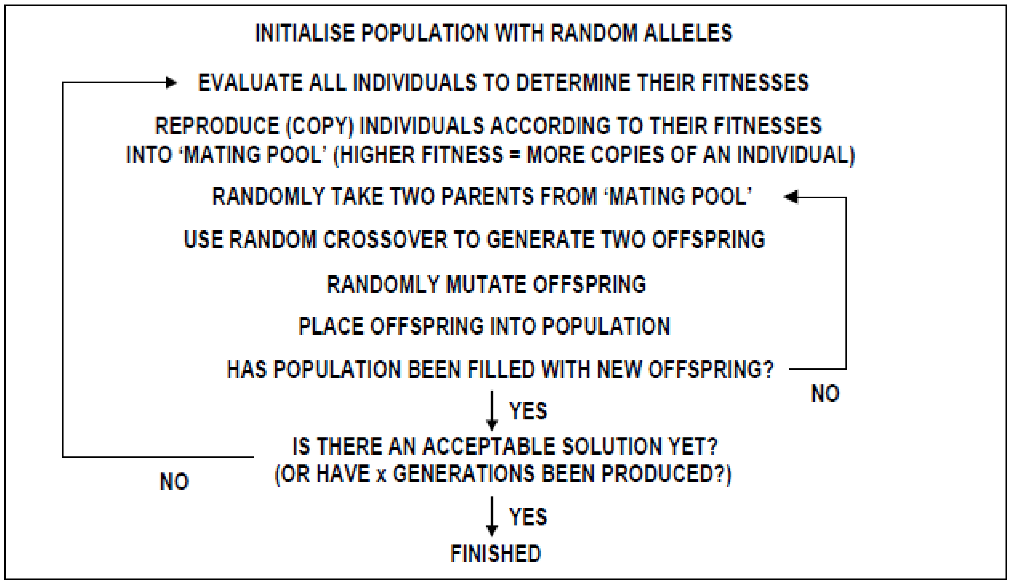

INITIALISE POPULATION WITH RANDOM ALLELES
EVALUATE ALL INDIVIDUALS TO DETERMINE THEIR FITNESSES
REPRODUCE (COPY) INDIVIDUALS ACCORDING TO THEIR FITNESSES INTO 'MATING POOL' (HIGHER FITNESS = MORE COPIES OF AN INDIVIDUAL)
RANDOMLY TAKE TWO PARENTS FROM 'MATING POOL'
USE RANDOM CROSSOVER TO GENERATE TWO 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 3. Genetic algorithm

Genetic programming is a specialized form of genetic algorithm. It follows essentially the same procedure as genetic algorithm. The difference between these two algorithms is:

1. In genetic programming algorithm, genotypes are the same as phenotypes. That means genetic programming algorithm does not manipulate coded versions of the solutions, it manipulates the solutions themselves [14].

2. Genetic programming algorithm uses a hierarchical tree structure to represent the solutions, which can make a variable-length for each solution. In this way, the crossover process can be used to interchange randomly chosen branches of the parents' trees without the syntax of the programs being disrupted. As shown by Figure 4 [14].

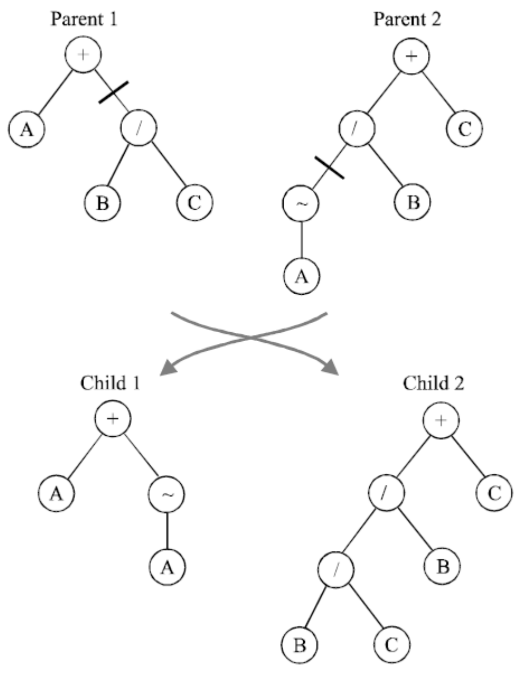


Figure 4. The behavior of the crossover operator in GP

Evolution strategies were initially designed with the goal of solving difficult discrete and continuous, parameter optimization problems [19]. Figure 5 shows the operation of the evolutionary strategy [14].

Step 1: The evolution strategies algorithm is initialized with a population of random solutions, or from a population of solutions mutated from a single solution provided by the user.

Step 2: Choose parent solutions from these populations.

Step 3: Use random recombination to generate offspring and place them into the 'child population', until enough new individuals fill the child population.

Step 4: Mutate each offspring within the 'child population'.

Step 5: Evaluate each individual by fitness measures. Select the fittest offspring and place them into the parent population.

Step 6: Evolution terminates after a predefined number of generations, or when a solution of sufficient quality has been generated.

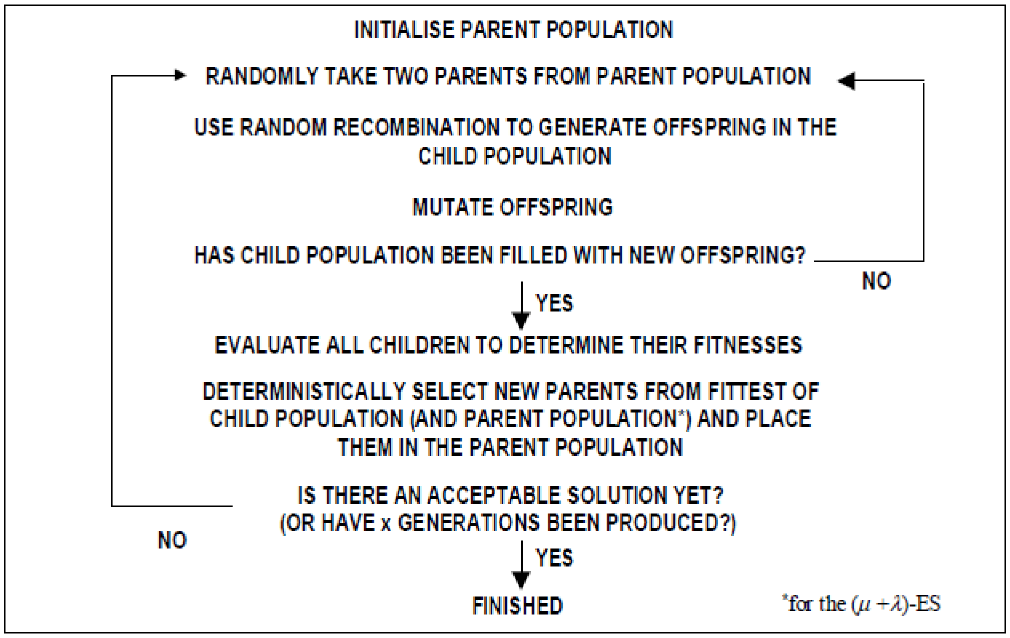

INITIALISE PARENT POPULATION
RANDOMLY TAKE TWO PARENTS FROM PARENT POPULATION
USE RANDOM RECOMBINATION TO GENERATE OFFSPRING IN THE CHILD POPULATION
MUTATE OFFSPRING
HAS CHILD POPULATION BEEN FILLED WITH NEW OFFSPRING?
NO
YES
EVALUATE ALL CHILDREN TO DETERMINE THEIR FITNESSES
DETERMINISTICALLY SELECT NEW PARENTS FROM FITTEST OF CHILD POPULATION (AND PARENT POPULATION*) AND PLACE THEM IN THE PARENT POPULATION
IS THERE AN ACCEPTABLE SOLUTION YET? (OR HAVE x GENERATIONS BEEN PRODUCED?)
NO
YES
FINISHED
*for the ($\mu + \lambda$)-ES

Figure 5. Evolution strategies

Evolution programming resembles evolutionary strategies closely (as shown in Figure 6). The difference between these two algorithms is:

1. Evolution programming uses an asexual reproduction to generate child population. All child population is generated by mutating a randomly chosen parent individual. There are no crossover or recombination processes in evolution programming [14].

2. In evolution strategy algorithms, new offspring can only reproduce from the best two parents of a previous generation. On the other hand, evolution programming

allowed the algorithm to reproduce new offspring from the better half of the 'parent pool' [14].

3. In the selection process, evolution strategies only keep the two most fit individuals and place them into the parent population. Evolution programming keeps half of the population with the highest fitness and uses them for a new round of asexual reproduction [14].

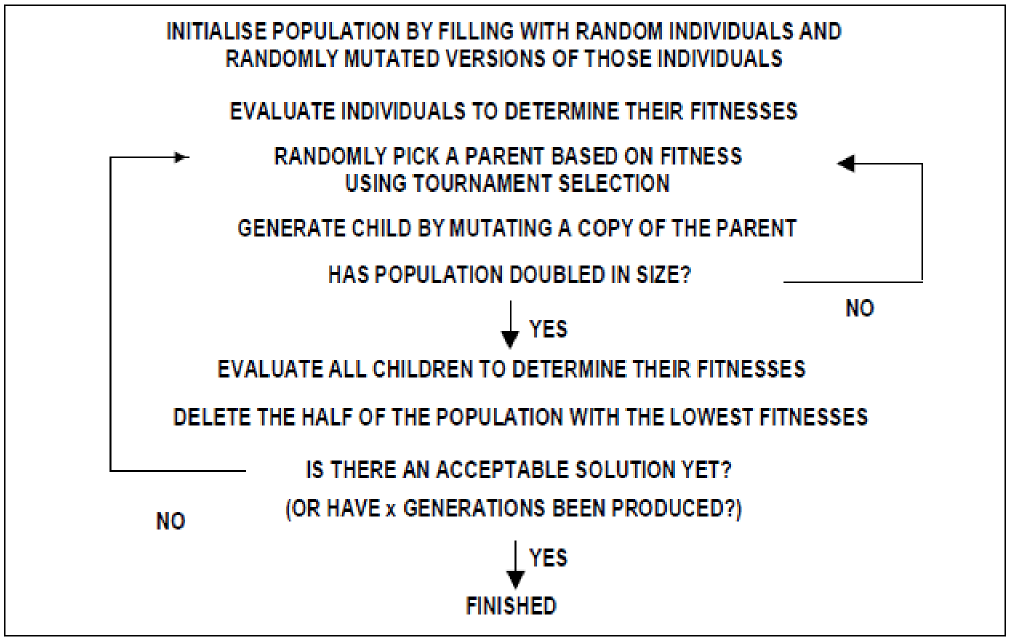

INITIALISE POPULATION BY FILLING WITH RANDOM INDIVIDUALS AND RANDOMLY MUTATED VERSIONS OF THOSE INDIVIDUALS
EVALUATE INDIVIDUALS TO DETERMINE THEIR FITNESSES
RANDOMLY PICK A PARENT BASED ON FITNESS USING TOURNAMENT SELECTION
GENERATE CHILD BY MUTATING A COPY OF THE PARENT
HAS POPULATION DOUBLED IN SIZE?
NO
YES
EVALUATE ALL CHILDREN TO DETERMINE THEIR FITNESSES
DELETE THE HALF OF THE POPULATION WITH THE LOWEST FITNESSES
IS THERE AN ACCEPTABLE SOLUTION YET? (OR HAVE x GENERATIONS BEEN PRODUCED?)
NO
YES
FINISHED

Figure 6. Evolution programming

2.3 Creative Evolutionary System

A creative evolutionary system is a system that uses some kind of evolutionary algorithm to explore creative solutions. This kind of system is designed to:

1) aid our own creative processes, and/or

2) to generate results to problems that traditionally required creative people to find solutions [12].

A creative evolutionary system framework is constructed by five elements (as shown in Figure 7):

1)An evolutionary algorithm

2)A genetic representation

3)An embryogeny using components

4)A phenotype representation

5)Fitness function(s) and/or processing of user input [12]

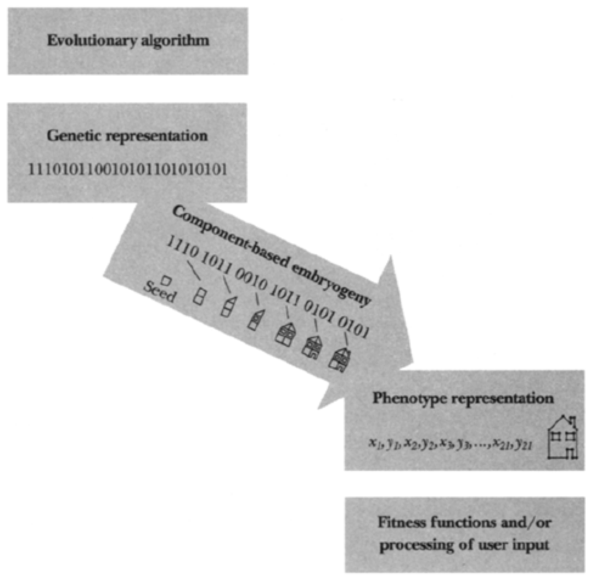

Evolutionary algorithm
Genetic representation
111010110010101101010101
Component-based embryogeny
1110 1011 0010 1011 0101 0101
Seed
Phenotype representation
$x_1, y_1, x_2, y_2, x_3, y_3, \ldots, x_{21}, y_{21}$
Fitness functions and/or processing of user input

Figure 7. The five elements of the framework for creative evolutionary systems

2.4 Evolutionary Computation Case (Engineering Design)

The use of genetic algorithms is now a well-established technique in engineering design applications. In the field of design, there are many pioneers who have made a lot of attempts, John Frazer is one of them. In 1992, John Frazer and his students used genetic algorithm in yacht hull design [20]. Their program was intended to explore a range of widely different alternative solutions to yacht design (Figure 8).

The program starts by generating a randomized initial population of yacht hull designs. All designs will be evaluated by a fitness function, which considered both objective and subjective factors. The objective engineering factors include parameters such as stability, center of buoyancy, wetted surface area, prismatic coefficient, and blocking. The subjective factors (i.e. designer criteria) mainly considered aesthetic appearance, historic tradition, and allusion of form.

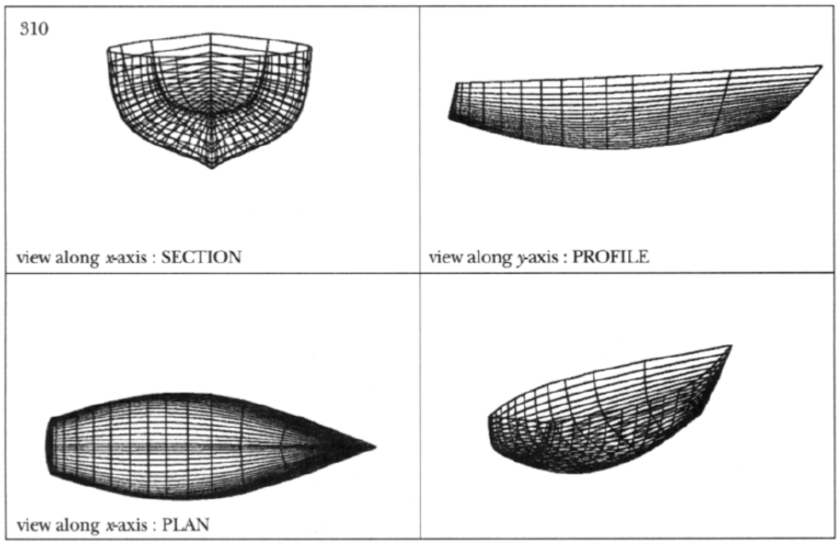


Figure 8. Yacht hulls

On the basis of these fitness functions, the designs with the highest fitness scores are selected and become the parent population. The crossover and mutation process is then applied to the parent population and produces new offspring. These processes repeat round and round until an excellent design appears.

The yacht hulls design program produces a range of choices for clients and provides a series of new ideas in yacht hulls design.

2.5 Evolutionary Computation Case (Music)

Music is a highly creative and varied thing. To compose music requires skill, imagination, and empathy. From contemporary music to jazz, there are many explorers trying the evolutionary method to generate music.

Vox Populi is an application that uses evolutionary computation to explore new musical structures [21]. The general processes of Vox Populi are shown in Figure 9.

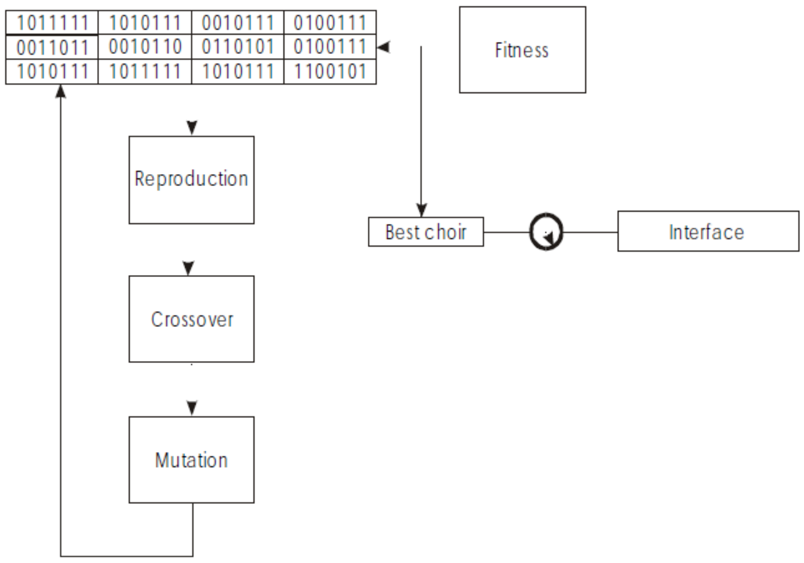


Figure 9. The genetic cycle of Vox Populi program

The individuals of the population consist of four voices which are randomly generated. Once the initial population of individuals has been created, the fitness of each individual is evaluated. The fitness function (F) is defined as a composition of three sub-functions: the harmonic fitness (H), the melodic fitness (M) and the voice range fitness (O). The value of these sub-functions is determined by the physical measurement of each voice.

$$F(O, M, H) = O(x1, x2, x3, x4) + M(x1, x2, x3, x4) + H(x1, x2, x3, x4)$$

After the fitness evaluation, typical operations of genetic programming like crossover and mutation are applied to the individuals. Then those individuals with higher fitness will be retained and treated as the parents of the next generation. This process will be repeated until the best chord is found. The following steps are realized in the genetic cycle:

STEP 1: Create an initial population randomly;

STEP 2: Until the termination criterion has been satisfied, perform the following:

1) Evaluate the fitness of each individual in the population;

2) Apply the genetic operators to individual chromosomes, or groups of voices, chosen with a probability based on fitness, to create a new population. That is:

· **Reproduction:** Copy existing individual strings to the new population;

· **Crossover:** Create two new chromosomes by crossing over randomly chosen sub-lists (substrings) from two existing chromosomes;

· **Mutation:** Create new chromosomes from an existing one by randomly mutating the character in the list.

STEP 3: Designate the best individual that appeared in any generation as the result.

2.6 Evolutionary Computation Case (Art)

Compared with music and design, art is more subjective, fuzzy and difficult to analyze. But it can still be evolved through evolutionary computing. One of the best known "evolutionary artists," Steven Rooke, uses genetic programming algorithm to evolve some astonishing pieces of art [22].

Rooke's image evolution system begins with a series of randomly generated tree structure genomes (as shown in Figure 10). Base on these genomes, the computer generates a corresponding image according to a specific mapping rule. In each generation, the image evolution system determines the parent genomes by random selection or user selection. The parent genomes generate new offspring through crossover and mutation steps.

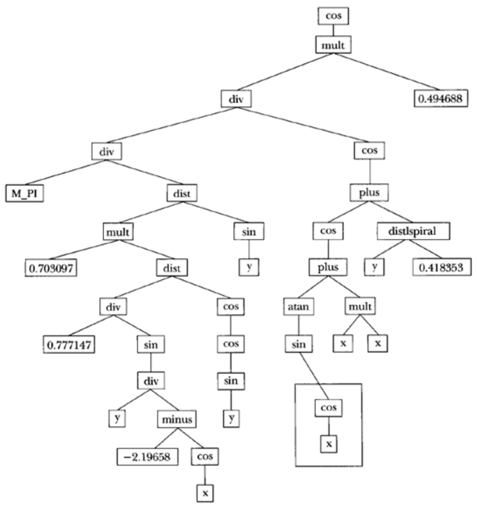


Figure 10. Tree structure genomes

In this way, Rooke's image evolution system generates a lot of creative images. Figure 11 shows some of his works.

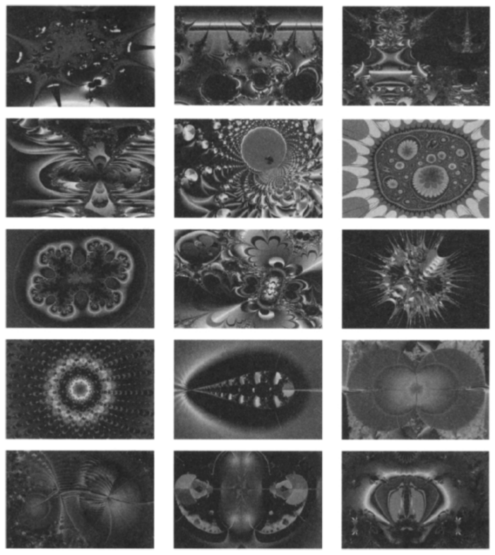


Figure 11. Creative images

CHAPTER 3

METHODOLOGY

3.1 A Framework for the Violin Explorer

The violin explorer is designed as a creative evolutionary system (mentioned in Chapter 2.3) which can help ordinary people to create their favorite violin design. Sections 3.1.1 to 3.1.5 illustrate five key elements of the framework for the violin explorer as a creative evolutionary system.

3.1.1 An Evolutionary Algorithm

A creative evolutionary system requires some kind of evolutionary algorithm to generate new solutions [14]. The majority of current implementations of evolutionary algorithms descend from these four approaches: genetic algorithms, evolutionary programming, evolution strategies, and evolution programming. The evolutionary algorithm of the violin explorer should be selected from these four algorithms based on actual demand.

Based on the demand of the violin explorer, it is more efficient to use a real number genotype instead of a binary parameter to represent the violin outline. This method will remove the mapping steps, which greatly reduces the computational strength and improves efficiency. From this perspective, the genetic algorithm is not the first choice for the violin explorer. In addition, the violin has evolved over centuries and the current size of violins conforms to ergonomic design. Also, based on the drawing principle of Digital Amati, the genotype of the violin explorer should be a fixed length. The genetic programming algorithm does not meet the need of the violin explorer. Finally, the violin explorer introduces human interaction processes to evaluate the violin outline. It is an exceptionally heavy task to let users' rank violin designs based on their personal preferences round by round. By contrast, it is a more reasonable choice for users to choose two of their favorite designs in each round.

Based on the analysis above, the evolution strategies algorithm is the most suitable evolutionary algorithm for the violin explorer.

3.1.2 A Genetic Representation

The genotype representation defines the search space of the evolutionary algorithm [14]. The genotype representation of the violin explorer should be a series of parameters that can manipulate AmatiML [10] (a domain-specific markup language used in Digital Amati software for drawing violin outlines) to evolve violin outline design. AmatiML can define a series of geometric elements such as points, lines, and circles, then connect these geometric elements in a specific order to create violin outlines. Users can draw different violins by changing the parameters of specific geometric elements. Here is an example of drawing a different violin outline by changing the parameters of Point q.

Figure 12 is a violin named "Original_Violin" which is drawn by AmatiML. It is formed by connecting geometric elements in a specific order. The AmatiML language defines the location of the point q, as shown in Figure 13. It shares the same Ycoordinate with Point Q and the X-coordinate of Point q is equal to the X-coordinate of Point A plus a distance of line segment XN/2.

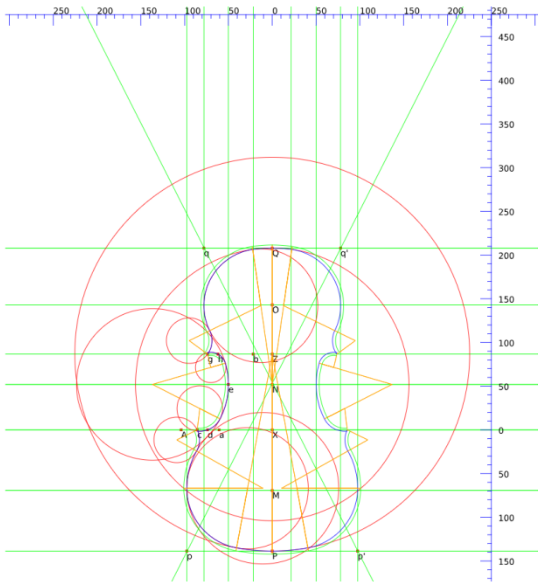


Figure 12. Original_Violin

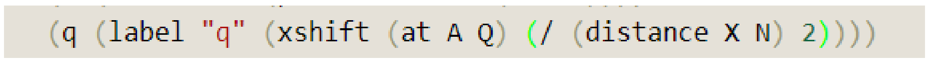

```
(q (label "q" (xshift (at A Q) (/ (distance X N) 2))))
```

Figure 13. The location of Point q

When users decide to change the location of point q, as shown in Figure 14. The new X-coordinate of Point q is equal to the X-coordinate of Point A plus a distance of the line segment XN/8. Correspondingly, a new violin design was created due to the influence of the new point q. The change of point q makes the upper bouts of the violin wider than previous designs, as shown in Figure 15.

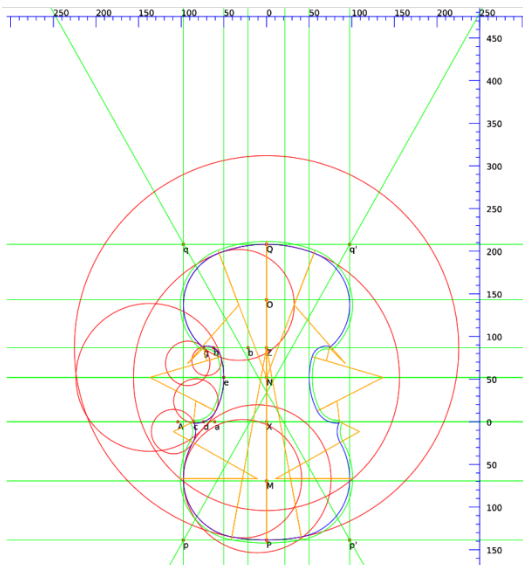
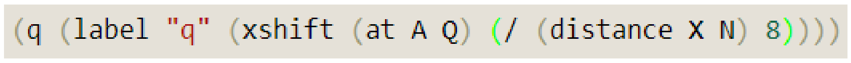


Figure 14. New_Violin



```
(q (label "q" (xshift (at A Q) (/ (distance X N) 8))))
```

Figure 15. New location of Point q

The above example shows how to draw different designs by changing parameters. Similarly, the violin explorer can change the outline of a violin by manipulating a series of parameters. This series of parameters will achieve a one-to-one mapping relationship with the violin outline which means there is a unique series of parameters for each violin outline. The collection of these series of parameters can be considered as the search space for violin design problems. Section 3.2.2 will provide parameters selection in more detail.

3.1.3 An Embryogeny Using Components

An embryogeny is a special kind of mapping process from genotype to phenotype [14]. The embryogeny process of the violin explorer will be implemented through

Digital Amati software. The Digital Amati software developed by Dr. Harry Mairson uses AmatiML language to define a series of geometric elements such as points, lines, and circles; and use a series of inscribed circles and reverse curves formed by these geometric elements to draw a violin outline.

3.1.4 A Phenotype Representation

Once the embryonic development process is over, the genotype representation will be transformed into a solution, which is defined as a phenotype representation. The phenotype representation of the violin explorer is a violin outline drawn by the Digital Amati software, as shown in Figure 16.

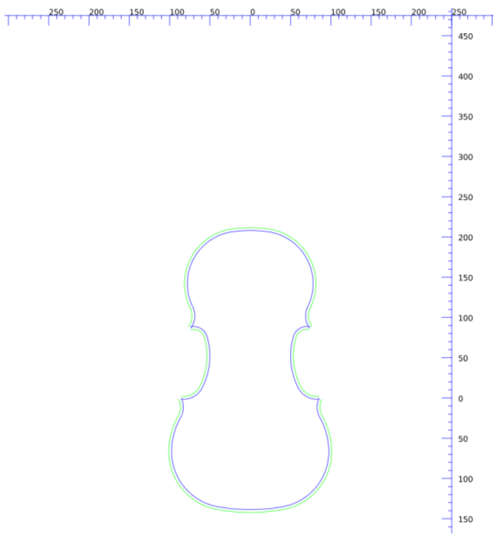


Figure 16. The phenotype representation of the violin explorer

These phenotype representation will be treated as the solution of a violin outline design and can be evaluated by either a fitness function or users.

3.1.5 Solution Evaluate

In this research, a violin outline design is considered as an aesthetic problem. It is almost an impossible task to establish a fitness function to quantify aesthetic standards into real values. Many factors such as users' age, background, and current fashion trends may become key factors influencing violin design. From this point of view, it is a more efficient and direct approach to introduce human interaction processes instead of using a fitness function to evaluate violin design. Users can directly integrate their imagination and creativity into violin design and create some novel solutions.

3.2 Algorithm of the Violin Explorer

The violin explorer uses evolutionary strategies algorithm to simulate a natural evolutionary process. These processes will help ordinary people design their favorite violin outline. The whole working processes are shown below (Figure 17):

Step 1: The violin explorer will initialize by randomly generating 10 violin outlines. These violin outlines will be treated as the original population pool.

Step2: Users are asked to select two of the most satisfying designs from the population pool.

Step3: These two designs will become the parents' population and will continually produce offspring through the recombination process. All offspring will be placed into the population pool until it reaches the maximum population (10 violin outlines).

Step4: A mutation process will be applied to each single violin outline in the population pool by using strategy parameters. The strategy parameters will also slightly mutate within the mutation process.

Step5: Once recombination and mutation have generated enough new individuals to fill the population pool, users will be asked to select two of the most satisfying designs from the population pool. These two designs will be treated as the parent population of the next generation.

Step6: Repeat Step 3 through Step 5 to generate new generations of design. These evolution processes will terminate after it reaches a predefined number of generations or when users are satisfied with the current design.

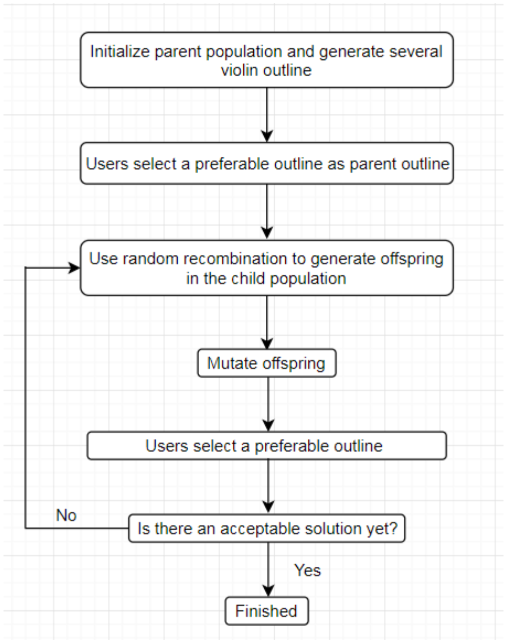

Initialize parent population and generate several violin outline
Users select a preferable outline as parent outline
Use random recombination to generate offspring in the child population
Mutate offspring
Users select a preferable outline
No
Is there an acceptable solution yet?
Yes
Finished

Figure 17. The whole processes of the violin explorer

3.2.1 Evolutionary Strategies Scheme

In each generation, the violin explorer will use specific ES schemes to continually generate violin designs to fulfill the population pool. The maximum individuals of the violin explorer population pool will set to a predefined number.

Currently, there are three major ES schemes: (1+1)-ES, (μ+ λ)-ES and (μ, λ)-ES (where μ is the number of parents and λ is the number of offspring). The (1+1)-ES scheme stands for a competition between one parent and one child. In this scheme, the maximum capacity of the population pool is 2, which will greatly reduce the computational strength and improve efficiency. Unfortunately, the (1+1)-ES scheme has a couple of drawbacks: these point-to-point competitions are easily stagnated at local optima, and it is extremely slow to converge on optimal solutions. As a modified version of (1+1) -ES, the (μ+ λ)-ES scheme improves the efficiency of search by increasing the scope of search in each generation. However, the (μ+ λ)-ES chooses to keep the old individuals (sometimes the local optimal solution), which may still lead to problems that tend to stagnate at the local optimum. Different from (1+1)-ES and (μ+ λ)-ES, the (μ, λ)-ES scheme avoids these drawbacks by slightly mutating each individual within the population pool.

Based on the analysis above, the violin explorer uses (μ, λ)-ES scheme to create new individuals in each generation. The violin explorer defined μ=2 and λ = 10, which means there are two parent populations that will be selected to create 10 offspring in each generation cycle. The maximum population of the population pool will be predefined as 10.

3.2.2 Genotype Representation Define

As mentioned in section 3.1.2, the genotype representation consists of an array of point locations. Each array corresponds to a violin outline in the search space.

Since the Digital Amati software is based on Euclidean geometry to draw the violin outline, each element in the genotype representation array has an dependent scope. For example, Point a & Point b are the center of two circles. Point g is the intersection of these two circles. We use Point g as the apex of the violin's upper left sharp corner (shown in Figure 18). As the position of Point a & Point b changes, the sharp corner of the violin changes. When the positions of Point a & b are far apart enough, the intersection (Point g) of the two circles disappears. Therefore, Point g has its own scope to ensure that the two circles intersect.

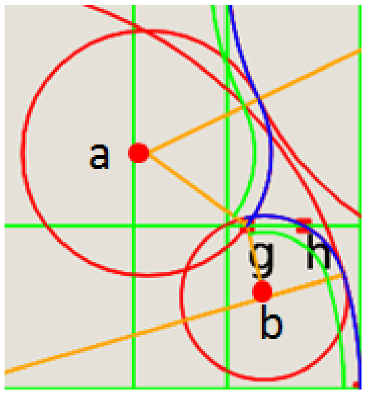


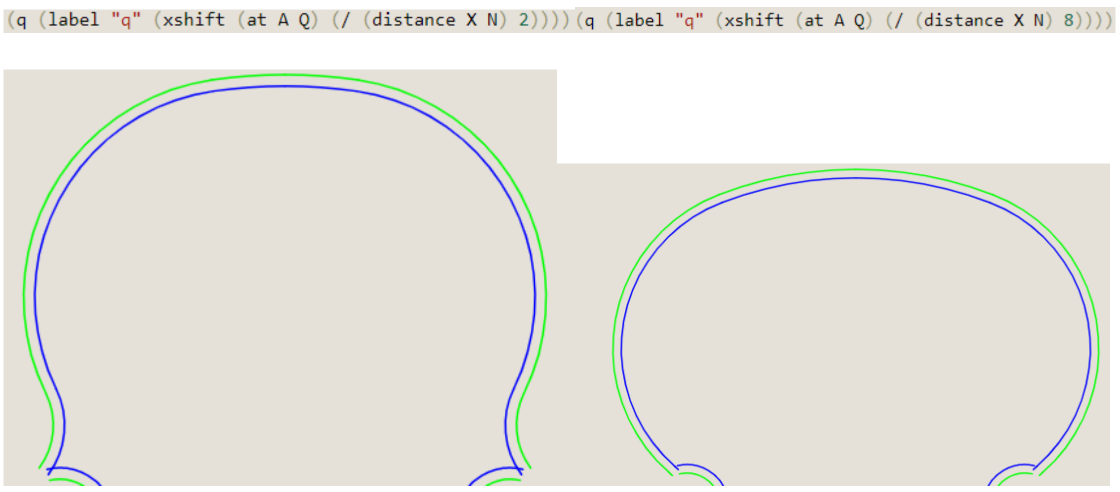
Figure 18. The scope of points

Meanwhile, for some points, their scopes are discontinuous. For example, the scope of Point P is 6-9 and 11-13. If we only choose one of them (scope 6-9 or scope 11-13), it will greatly reduce the search space. Technically, a large search space results in more potential solutions. The violin explorer expands the search space by adding 5 different models. Each model represents a genotype representation. Briefly, this method brings in five different models and runs the evolution strategies algorithm five times to achieve the expansion of search space. Table 1 summarize the different models chosen by the violin explorer.

Table 1. The violin explorer genotype representation models

Model No.	Genotype representation	Scope of each element
Model 1	[q,O,e,c]	q [1,8]/O [4,5]/e [6,10]/c [2,6]
Model 2	[q,O,e,c,P]	q [2,4] / O [3,5] / c [2,6] / P [6,9]/ e [8,10]
Model 3	[q,O,e,c,P]	q [2,4] / O [3,5] / c [2,6] / P [11,13] /e [8,10]
Model 4	[q,O,e,c,P]	q [5,8] / O [3,5] / c [2,6] / P [6,9] /e [8,10]
Model 5	[q,O,e,c,P]	q [5,8] / O [3,5] / c [2,6] / P [11,13] /e [8,10]

The value of Point q controls the width of the upper bouts of a violin. As the value of Point q increases, the upper bouts of violin will become wider (as shown in Figure 19).

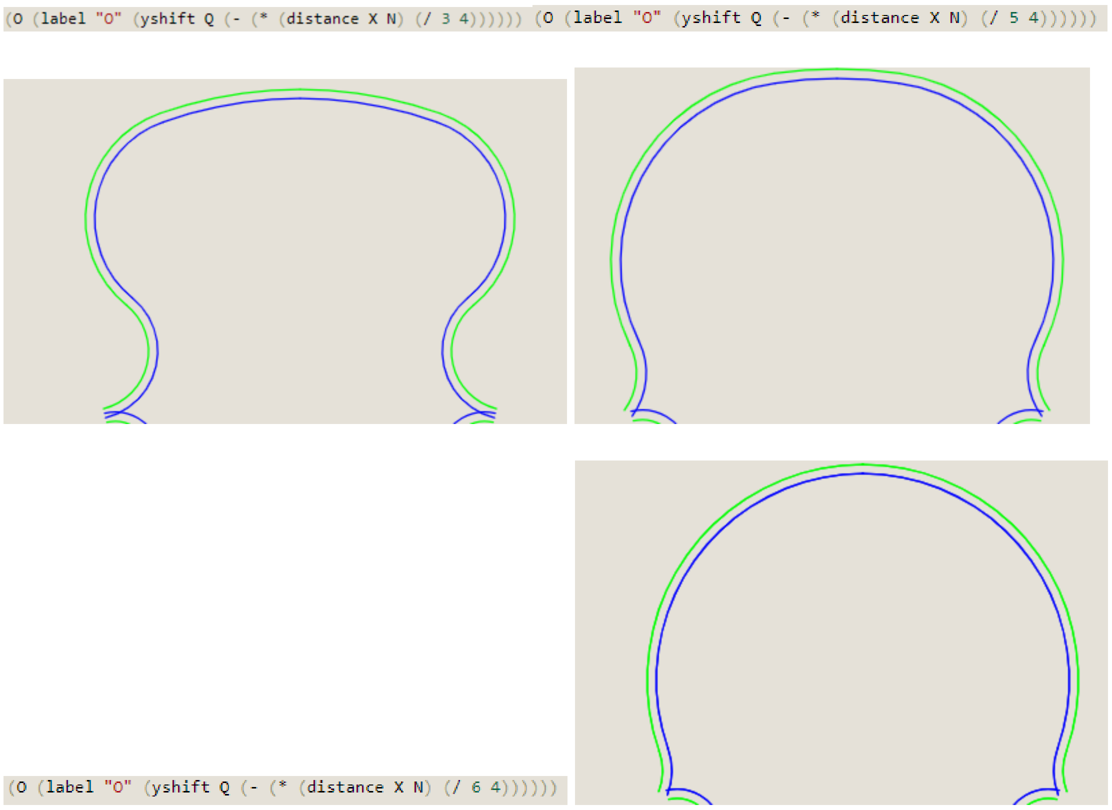


```
(q (label "q" (xshift (at A Q) (/ (distance X N) 2))))
```

```
(q (label "q" (xshift (at A Q) (/ (distance X N) 8))))
```

Figure 19. The effect of Point q value

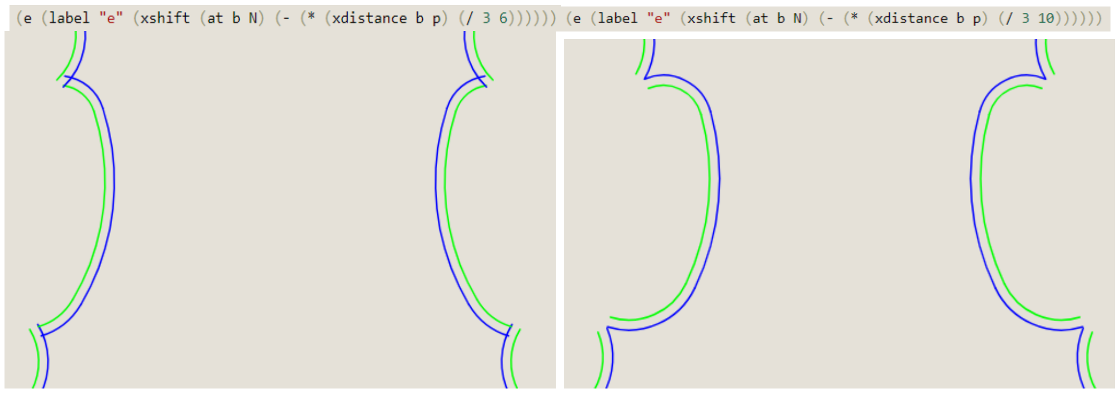
The value of Point O controls the radian of the upper bouts of a violin. When the value of Point O increases, the upper bouts of a violin is smoother. For example, there are three upper bouts design in Figure 20; the design on the left has the smallest value of Point O; the design on the middle is larger; the design on the right has the largest value of Point O.



```
(O (label "O" (yshift Q (- (* (distance X N) (/ 3 4))))))
(O (label "O" (yshift Q (- (* (distance X N) (/ 5 4))))))
(O (label "O" (yshift Q (- (* (distance X N) (/ 6 4))))))
```

Figure 20. The effect of Point O value

The value of Point e controls the width of the middle bouts of a violin. As the value of Point e increases, the middle bouts of the violin will become narrower (as shown in Figure 21).

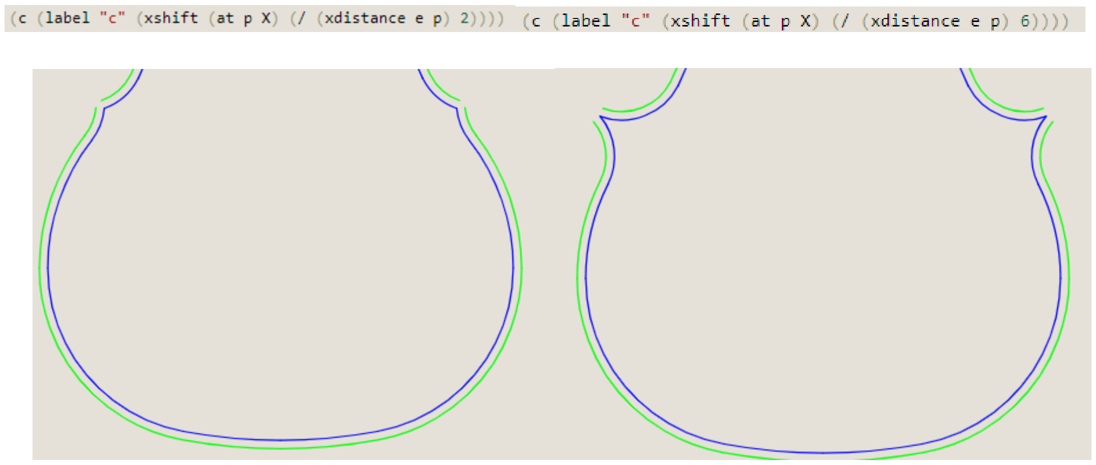


```
(e (label "e" (xshift (at b N) (- (* (xdistance b p) (/ 3 6))))))
```

```
(e (label "e" (xshift (at b N) (- (* (xdistance b p) (/ 3 10))))))
```

Figure 21. The effect of Point e value

The value of Point c controls the shape of the lower bouts. As the value of Point c increases, the corner of the violin's lower bouts become much sharper (as shown in Figure 22).

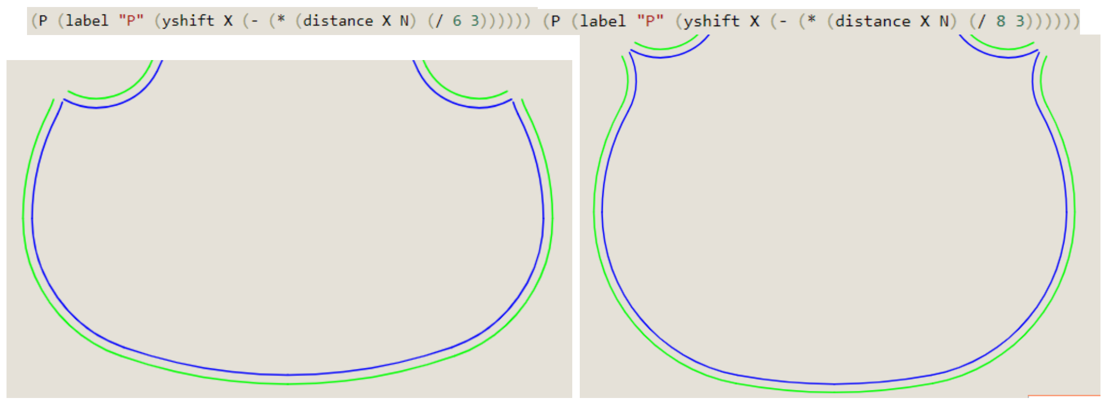


```
(c (label "c" (xshift (at p X) (/ (xdistance e p) 2))))
```

```
(c (label "c" (xshift (at p X) (/ (xdistance e p) 6))))
```

Figure 22. The effect of Point c value

The value of Point P affects the shape of a violin's lower bouts. As the value of Point P increases, the violin's lower bouts become taller and rounder (as shown in Figure 23).



```
(P (label "P" (yshift X (- (* (distance X N) (/ 6 3))))))
```

```
(P (label "P" (yshift X (- (* (distance X N) (/ 8 3))))))
```

Figure 23. The effect of Point P value

3.2.3 Recombination

The recombination process exists in each generation cycle. The violin explorer uses the recombination process to recombine parent populations and create new offspring. The violin explorer defined an integer "bp", which is randomly generated and designed to determine the position to break a genotype into two pieces. For example, when "bp" =3, both parents 1 & 2 will be divided into two pieces from the 3rd position. Then, the randomly generated number "rnd" will decide how to recombine these 4 pieces and create a new offspring by using these pieces (Figure 24).

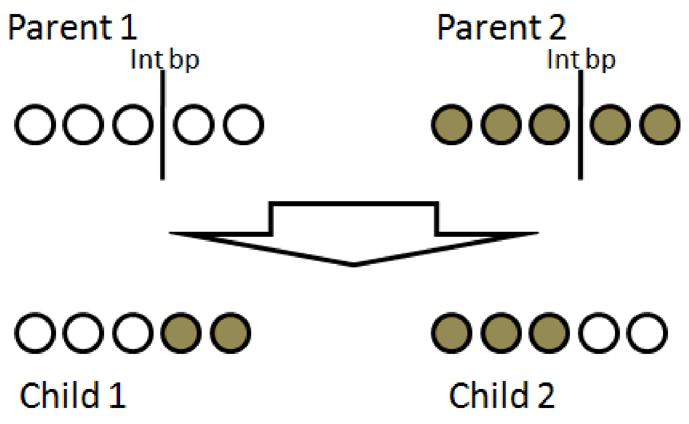
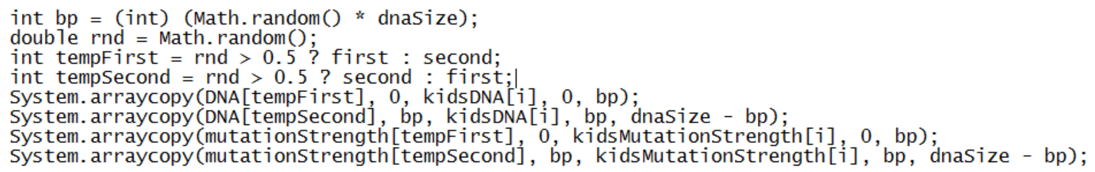


Figure 24. Recombination process

The violin explorer defined a randomly generated parameter "rnd", which is designed to selects a parent population to provides the first half "genes" while another parent population provides the second half "genes" to their offspring. For example, if "rnd" >0.5, Parent 1 will provide the first half "genes" and Parent 2 will provide the second half "genes" to create a new offspring. Otherwise, Parent 2 will provide the first half "genes" and Parent 1 will provide the second half "genes". As the code is shown in Figure 25.



```
int bp = (int) (Math.random() * dnaSize);
double rnd = Math.random();
int tempFirst = rnd > 0.5 ? first : second;
int tempSecond = rnd > 0.5 ? second : first;
System.arraycopy(DNA[tempFirst], 0, kidsDNA[i], 0, bp);
System.arraycopy(DNA[tempSecond], bp, kidsDNA[i], bp, dnaSize - bp);
System.arraycopy(mutationStrength[tempFirst], 0, kidsMutationStrength[i], 0, bp);
System.arraycopy(mutationStrength[tempSecond], bp, kidsMutationStrength[i], bp, dnaSize - bp);
```

Figure 25. Recombination process

The recombination process will continue to be repeated until sufficient offspring are generated.

3.2.4 Mutation

Mutation plays an important role in the evolutionary strategy and is regarded as the primary search operator [14]. The mutation process in the violin explorer involves making small additions or subtractions to each array element. The value of additions or subtractions is determined by the strategy parameter: mutation strength. Mutation strength is an array that is randomly generated when the original population is initialized. The mutation strength array has the same length as the genotype representation array, and each array element within mutation strength array will correspond to the array element of genotype representation. As shown in Figure 26, the mutation strength is used to produce random deviations according to a Gaussian distribution with a mean of zero. These random deviations will apply to each element of genotype representation, and make them slightly mutated.

```
kidsDNA[i][j] += num * kidsMutationStrength[i][j] * Math.random();
```

Figure 26. Mutate each element within the genotype representation

The random integer "num" will determine whether it is an addition or subtraction to each parameter (as shown in Figure 27).

```
int num = Math.random() > 0.5 ? 1 : -1;
```

Figure 27. Additions or subtraction

During the evolving process, the new individuals will become closer and closer to the optimal solution round by round. It is a good idea to reduce the step size generation by generation to reach the global optimum faster. The violin explorer uses a constant (0.9) to reduce the mutation strength generation by generation, as shown in Figure 28.

```
kidsMutationStrength[i][j] = mutationStrength[i][j] * 0.9;
```

Figure 28. Reduce the mutation strength

3.2.5 Selection

The selection process is extremely simple. The violin explorer has introduced human interactive processes to evaluate solutions. In each generation, users are asked to select two violin designs from the population pool. Their choice will become the parents of the next generation.

3.2.6 Generate Violin Images

In every evolve cycle, a new genotype representation will be produced through recombination and mutation processes. The violin explorer uses the elements within the new genotype representation array to replaces the original violin drawing code (as shown in Figure 29).

```
tempCode = tempCode.replace("{q}", Double.toString(es[model-1].getDNA()[i][0]));
tempCode = tempCode.replace("{O}", Double.toString(es[model-1].getDNA()[i][1]));
tempCode = tempCode.replace("{c}", Double.toString(es[model-1].getDNA()[i][2]));
tempCode = tempCode.replace("{P}", Double.toString(es[model-1].getDNA()[i][3]));
tempCode = tempCode.replace("{e}", Double.toString(es[model-1].getDNA()[i][4]));
```

Figure 29. Generate new violin drawing code

After the new violin drawing code is generated, the next step is to uses this code to generate a violin design. With the Retrofit 2 library [23], we can build an application programming interface and links to the violin explorer. Then, the violin explorer sends asynchronously request to API and get violin designs. The image generated by Digital Amati is in SVG format, which can not directly display. The violin explorer uses the Batik library [24] to solve this problem. With the Batik library, you can easily display an SVG document.

CHAPTER 4

Results

4.1 Experiments Setup

Designed as a creative evolutionary system, the violin explorer is expected to work creatively like a human designer. The violin explorers will be tested by an anonymous experiment to verify software usability. In addition, the experiments will explore whether users' with different ages and backgrounds can use violin explorer to design a novel violin outline.

This experiment will randomly invite 100 ordinary people to anonymously use the violin explorer. Considering the protection of personal privacy, no personal information such as their name, their age and their artistic ability will be recorded. All participants will be asked two questions to determine their experimental grouping:

1. Which age group are you in? "18 to 40" or "older than 40"?

2. Do you have an artistic background such as painting or musical instruments?

Based on the participants' answers to these two questions, they will be divided into four experimental groups, as shown by Table 2.

Table 2. Four experimental groups

Group 1	Participants are between the ages of 18 to 40 and have an artistic background.
Group 2	Participants are between the ages of 18 to 40 but do not have any artistic background.

Group 3	Participants' age is older than 40 and has an artistic background.
Group 4	Participants' age is older than 40 but does not have any artistic background.

After grouping, participants will be invited to use the violin explorer to design their favorite violin appearance. During the experiment, participants have the right not to answer the survey question, as well as to withdraw completely from the experiment at any point during the process. Additionally, participants have the right to request that the researchers not use any of their responses. After the experiment, the violin explorer will save participants' violin outline images for research purposes.

For each experiment group, the violin design parameters will be averaged to create a new violin design and see if there is any difference between them. In addition, these new violin designs will be compared with traditional violin designs to see if there are novel designs. Finally, some novel designs will be selected from these designs for display purposes.

4.2 Experiments Results

During the anonymous experiment, the usability of the violin explorer has been verified. Participants designed their violins according to their personal preferences. A lot of creative violin outlines were designed. Compared with the original violin design, these violin outlines have a series of changes (as shown in Figure 30).

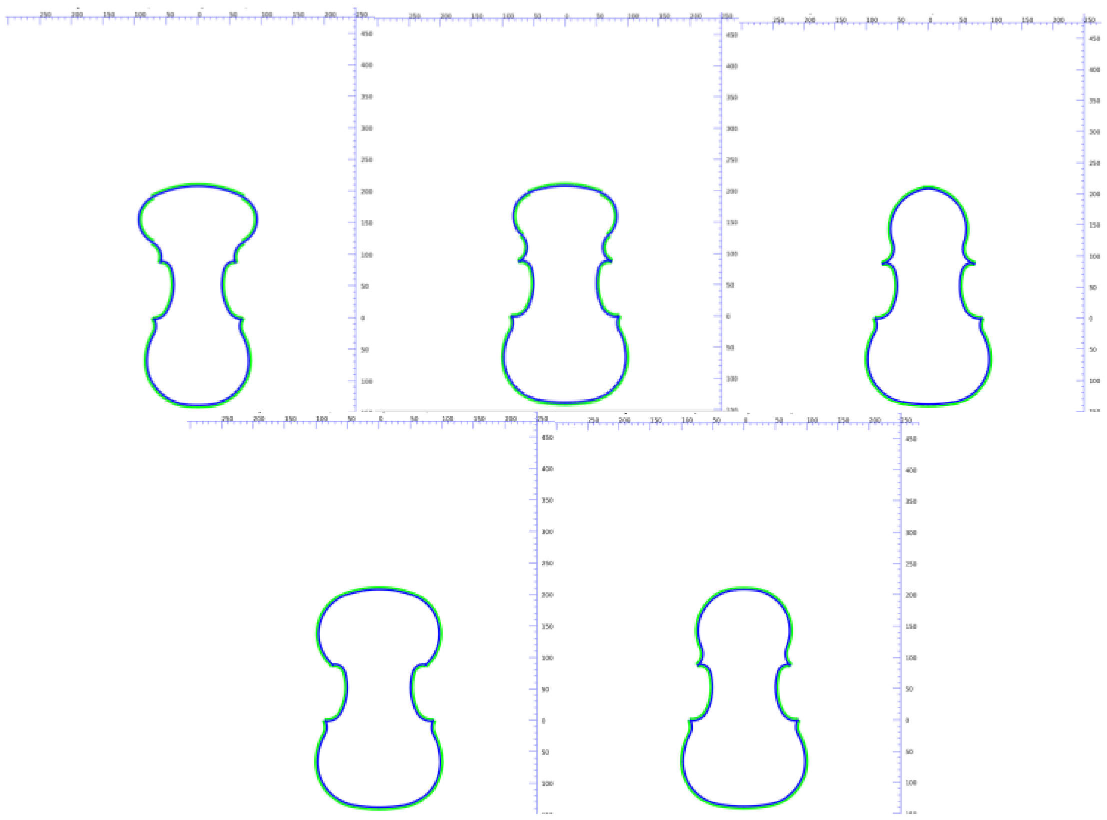


Figure 30. New violin outlines

4.2.1 Experiment Group 1

The participants in experiment Group 1 were between 18 and 40 years old and had an artistic background. Based on anonymous experiments, we collected 25 violin designs (as shown in Figure 31).

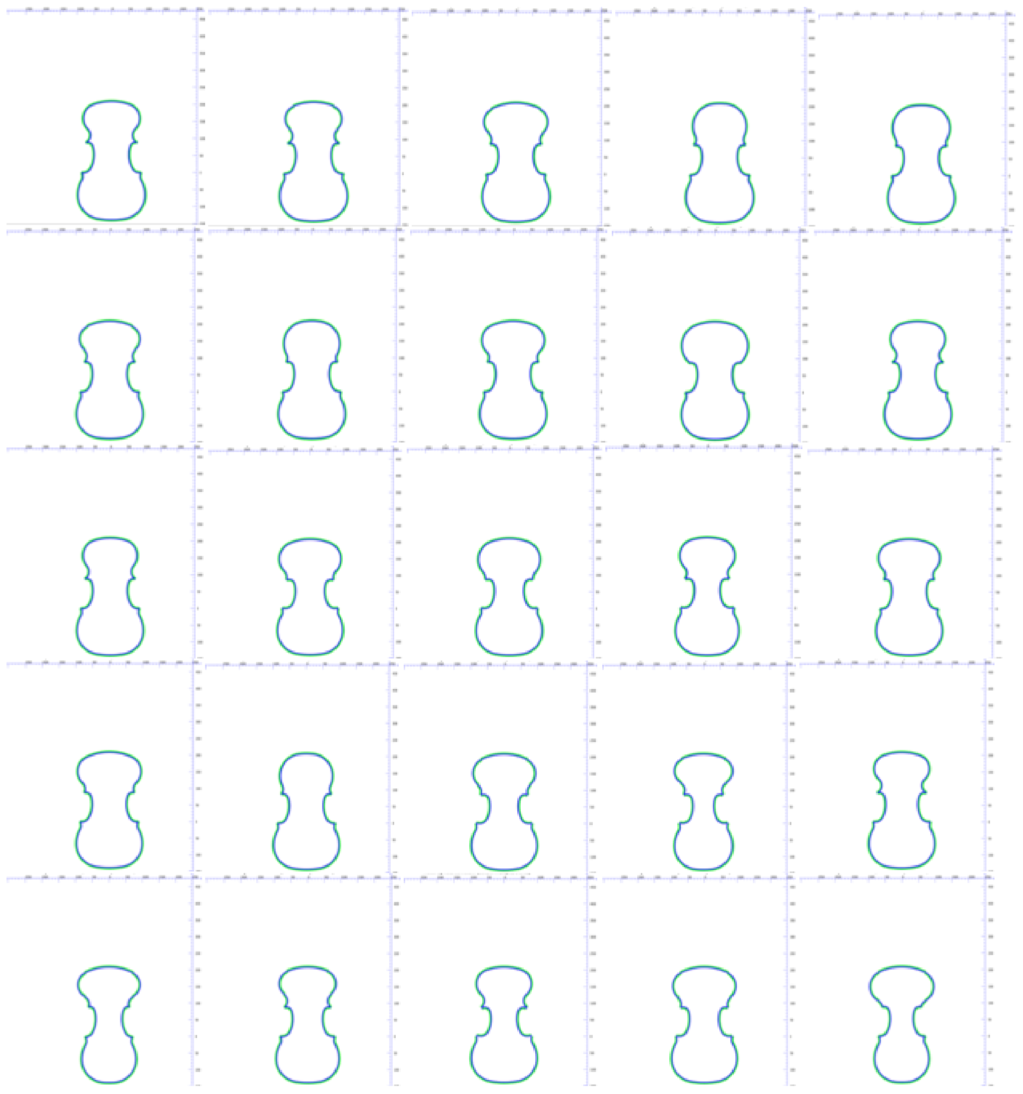


Figure 31. The designs of experiment Group 1

After we average the elements within Group 1's genotype representation array, the average outline of Group 1 is shown in Figure 32.

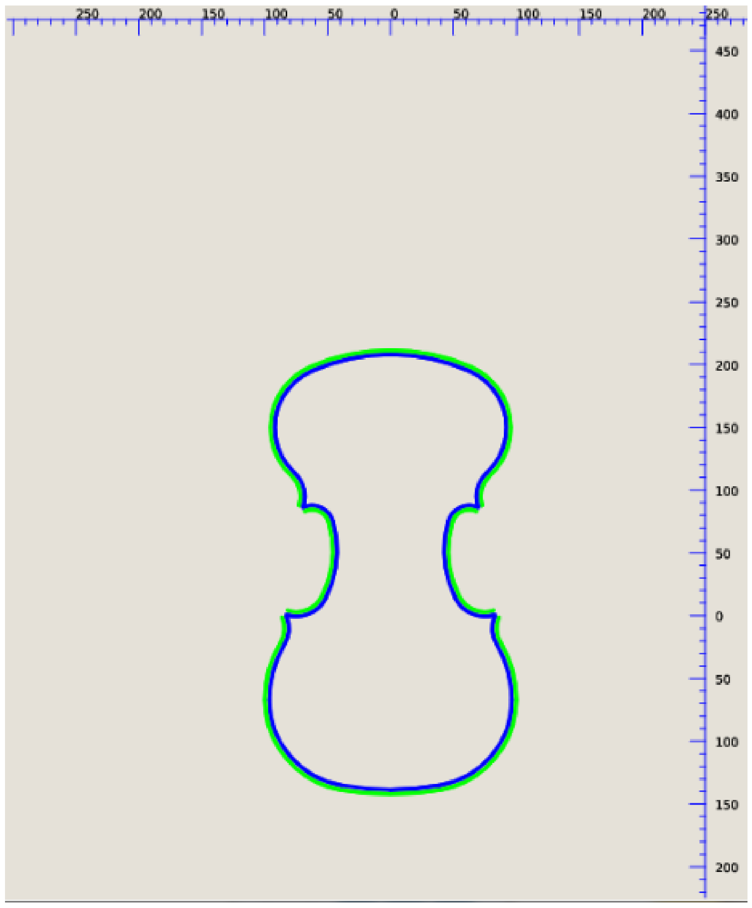


Figure 32. The average outline of Group 1

4.2.2 Experiment Group 2

The experiment Group 2 were between 18 and 40 years old and had no artistic background. Based on anonymous experiments, we collected 25 violin designs (as shown in Figure 33).

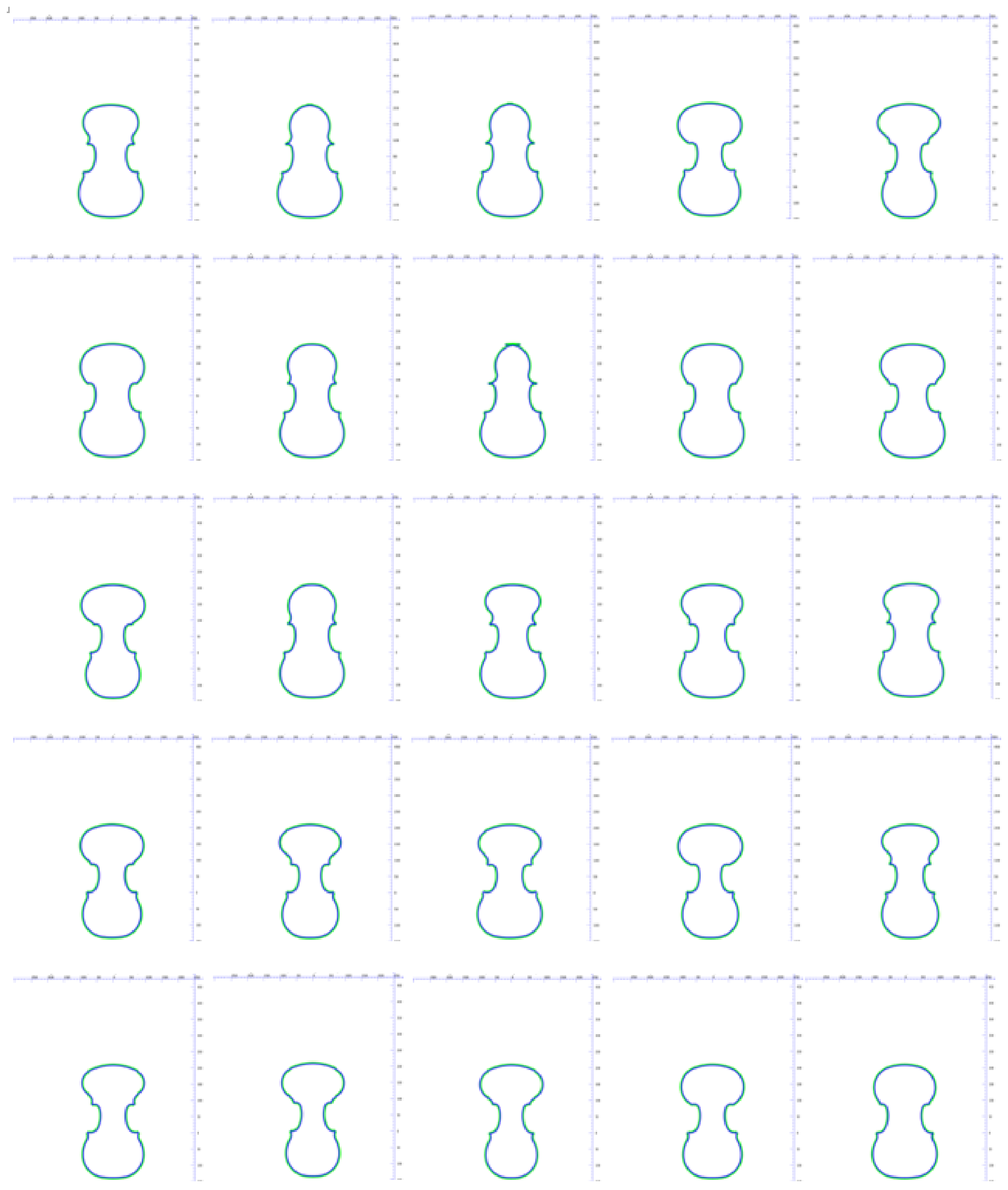


Figure 33. The designs of experiment Group 2

After we average the elements within Group 2's genotype representation array, the average outline of Group 2 is shown in Figure 34.

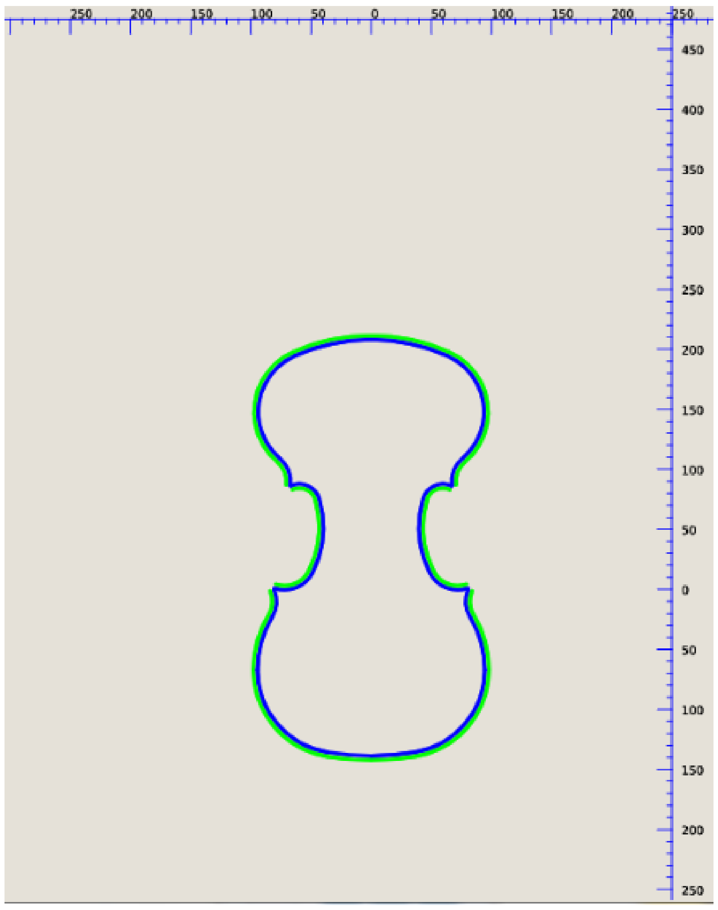


Figure 34. The average outline of Group 2

4.2.3 Experiment Group 3

The participants in experiment Group 3 were older than 40 years old and had an artistic background. Based on anonymous experiments, we collected 25 violin designs (as shown in Figure 35).

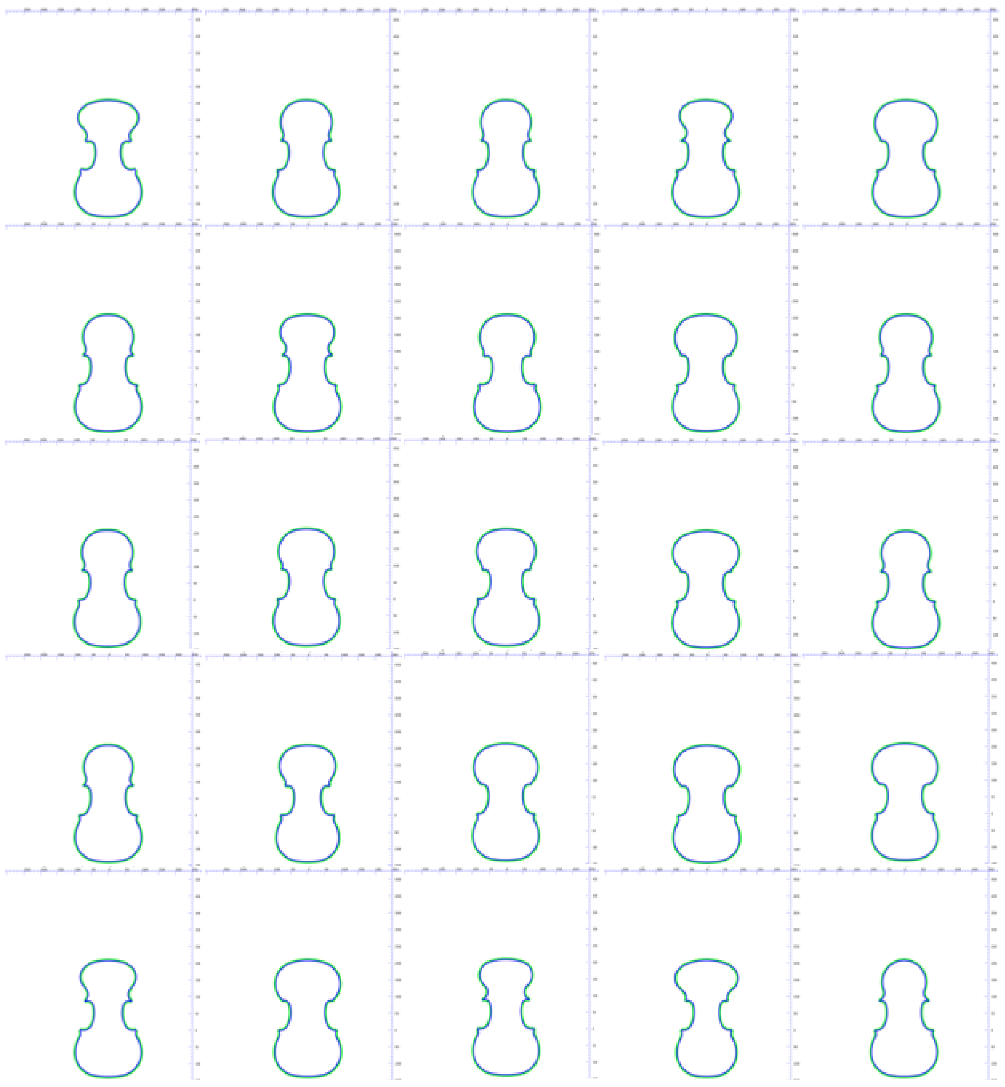


Figure 35. The designs of experiment Group 3

After we average the elements within Group 3's genotype representation array, The average outline of Group 3 is shown in Figure 36.

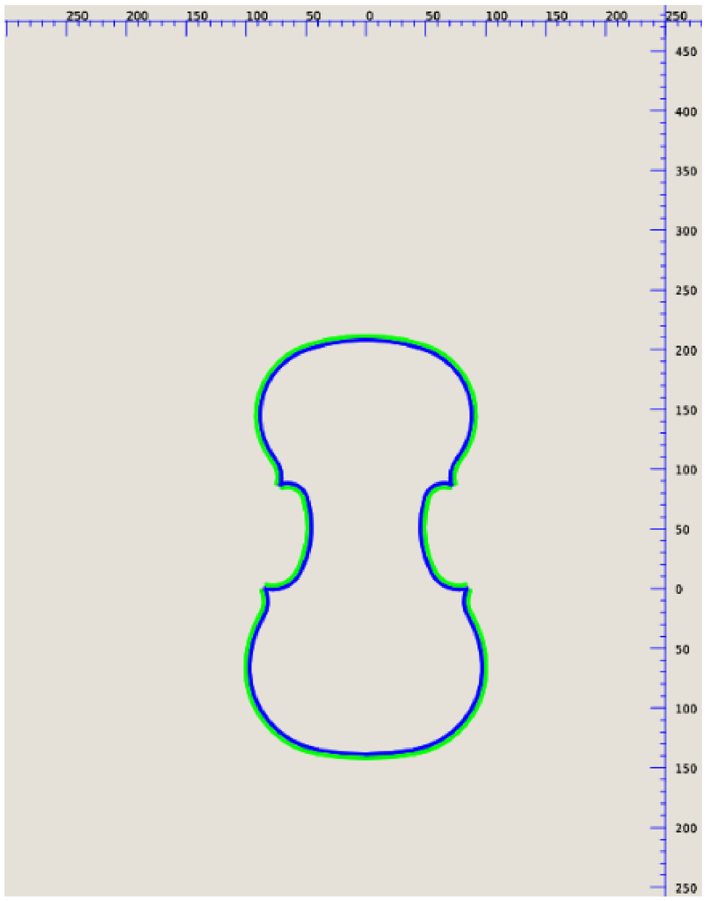


Figure 36. The average outline of Group 3

4.2.4 Experiment Group 4

The participants in experiment Group 4 were older than 40 years old and had no artistic background. Based on anonymous experiments, we collected 25 violin designs (as shown in Figure 37).

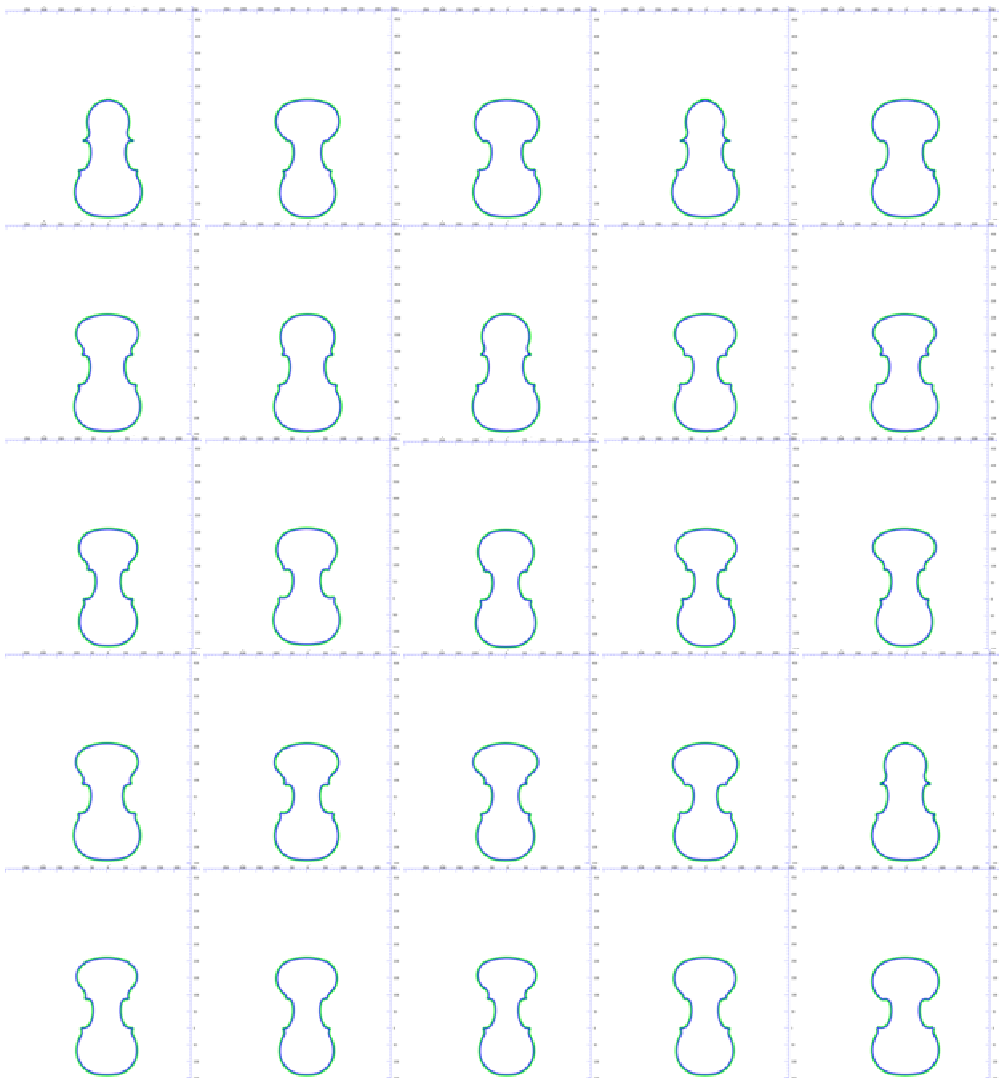


Figure 37. The designs of experiment Group 4

After we average the elements within Group 4's genotype representation array, The average outline of Group 4 is shown in Figure 38.

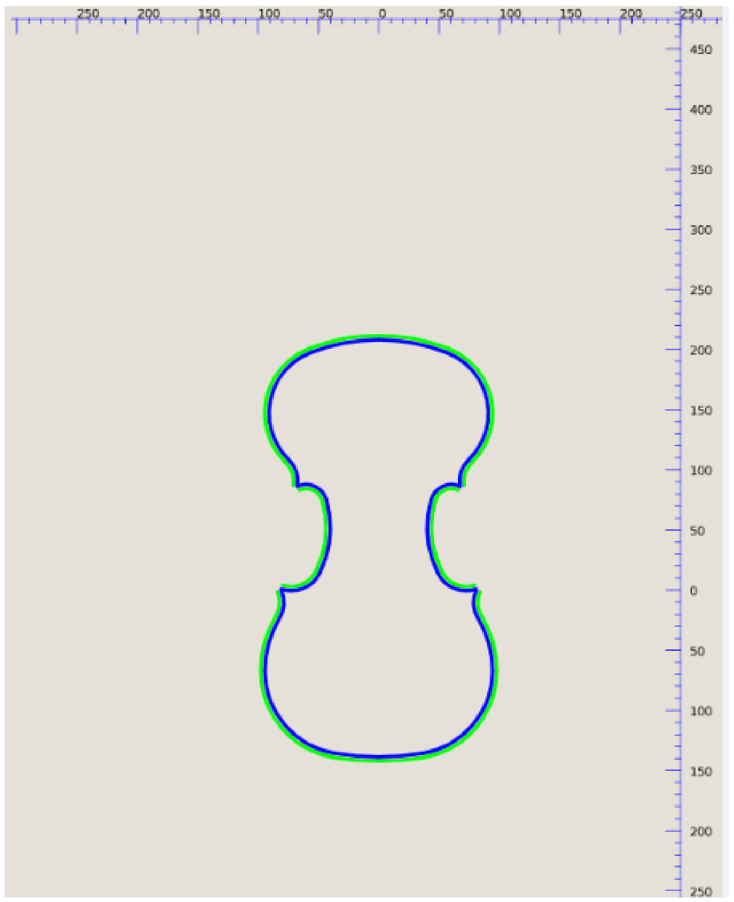


Figure 38. The average outline of Group 4

4.3 Results Compare

The average genotype representation of each experimental group is shown in Table 3. By comparing this to the standard violin design, we can see how far away the means for different groups are from the "standard" violin design (as shown in Table 4). Although the genotype representation of each group is different, we still can see some trends from it.

Table 3. The average genotype representation of each group

	q	O	c	P	e
Standard design	2	5	2	8	8
G1	4.348	3.788	3.568	6.832	8.28
G2	5.112	3.992	3.648	5.42	8.84
G3	3.336	4.496	2.796	7.508	8.144
G4	3.988	4.156	3.8	5.38	8.588

Table 4. The difference between each group and standard design

	q	O	c	P	e
Standard design	2	5	2	8	8
Difference between G1 and Standard design	2.348	-1.212	1.568	-1.168	0.28
Difference between G2 and Standard design	3.112	-1.008	1.648	-2.58	0.84
Difference between G3 and Standard design	1.336	-0.504	0.796	-0.492	0.144
Difference between G4 and Standard design	1.988	-0.844	1.8	-2.62	0.588

Compared with the "standard" violin design, four experimental groups have some similar trends:

(1) Participants in all four experimental groups tended to choose wider upper and lower bouts.

(2) Participants in all four experimental groups prefer a sharper violin lower bouts corner with shorter lower bouts.

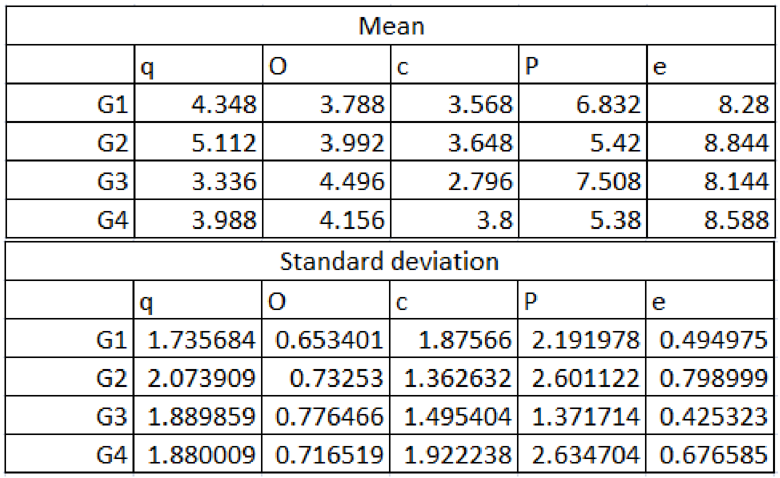
(3) Also, a more curvilinear upper bout is preferred.

(4) The width of the middle bouts is narrower but closed to the original design.

Overall, the average violin outlines of the four experimental groups are a little similar to each other. This result may be caused by many factors. First, using the average method might eliminate some extreme solutions. These extreme solutions may be the most characteristic solutions. Second, the experiment only invites 100 people, which is a very small sample size. Third, participants with a centralized geographical location may also lead to a more similar aesthetic.

After the average shape comparison, I want to see if there is any further difference between the shapes of four experimental groups. According to the mean and standard deviation of each group (shown in Table 5), I calculate the 1,2,3 standard deviation to see if there is any difference between each group. By using this method, only the shapes at 1 standard deviation are within the scope of the violin's genotype representation.

Table 5. The mean and standard deviation of each group



Mean					
	q	O	c	P	e
G1	4.348	3.788	3.568	6.832	8.28
G2	5.112	3.992	3.648	5.42	8.844
G3	3.336	4.496	2.796	7.508	8.144
G4	3.988	4.156	3.8	5.38	8.588

Standard deviation					
	q	O	c	P	e
G1	1.735684	0.653401	1.87566	2.191978	0.494975
G2	2.073909	0.73253	1.362632	2.601122	0.798999
G3	1.889859	0.776466	1.495404	1.371714	0.425323
G4	1.880009	0.716519	1.922238	2.634704	0.676585

The shapes at 1 standard deviation away from the mean are drawn at Figure 39 and Figure 40. The shapes at 1 standard deviation (μ-σ) shows us the younger participants (18-40) prefer a wider upper bout. The design of participants in Group 3 are similar to the standard violin design.

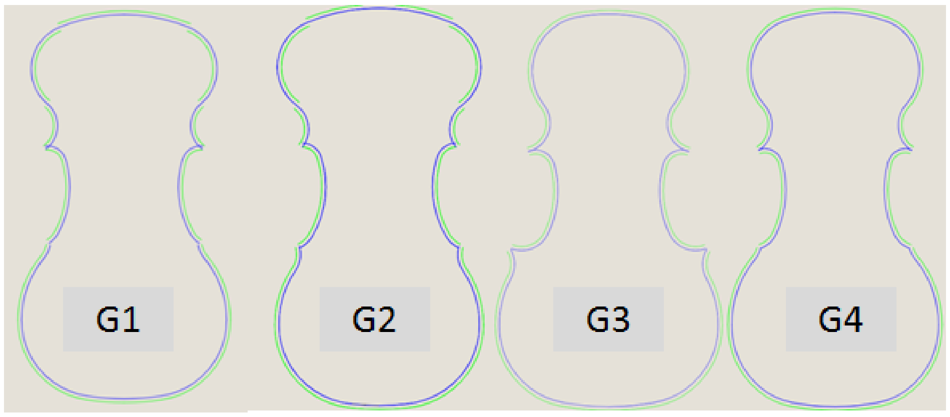


Figure 39. Shapes at 1 standard deviation (μ-σ)

The shapes at 1 standard deviation (μ+σ) did not show much difference, the shapes of four experimental groups are very similar.

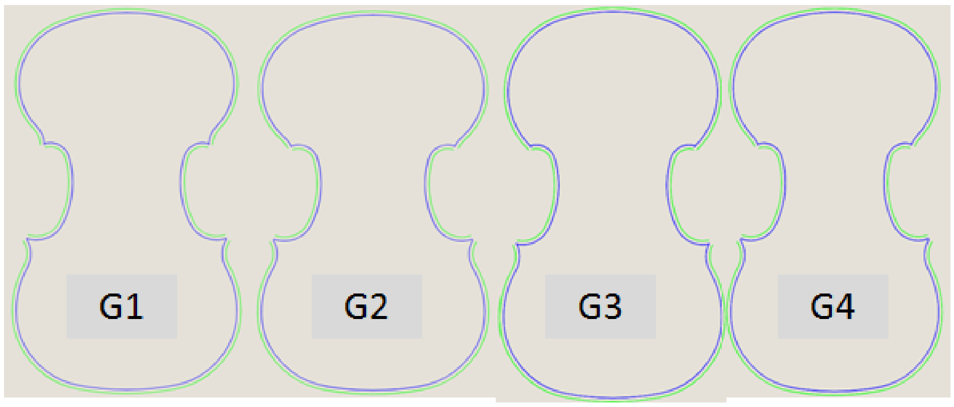


Figure 40. Shapes at 1 standard deviation (μ+σ)

There is another interesting trend in this experiment. The participants with an artistic background have a lower standard deviation between their design and the standard violin design. Also, the older participants have a lower standard deviation between their design and the original violin design. The older participants with an artistic background have the lowest standard deviation, which means their designs are the most closed to the original violin design (as shown in Figure 41 and Table 6).

Table 6. The average of the standard deviation of each group

STDEV	q	O	c	P	e	average
G1	2.899172	1.370693	2.415781	2.444749	0.56	1.938079
G2	3.716665	1.237417	2.120943	3.626513	1.151173	2.370542
G3	2.283331	0.912579	1.667453	1.431223	0.440908	1.347099
G4	2.710203	1.097816	2.605226	3.678097	0.886115	2.195491

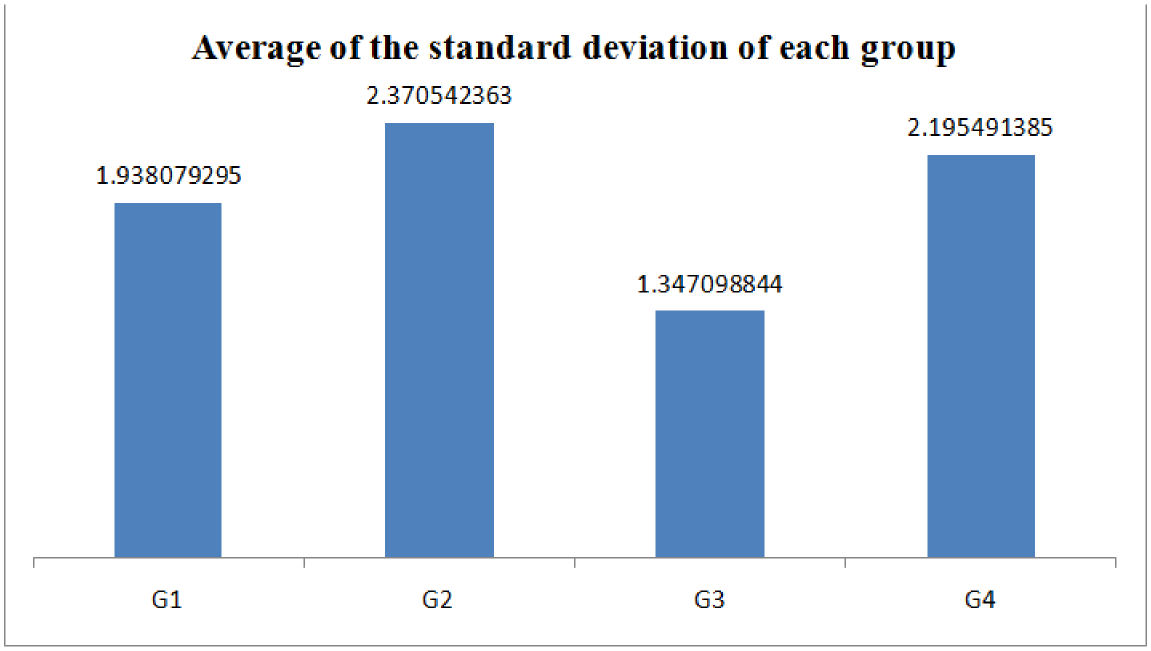


Figure 41. The average of the standard deviation of each group

This trend reveals that there might be a potential connection between the age and background of the participants and their designs. The older people and those people with an artistic background might prefer a "classic" violin design. The younger people are more open-minded in designing their favorite violin.

The Figure 42 shows the average shape of four groups' genotype representation. We'll see the difference between the underlying theme of violin outline and the original violin.

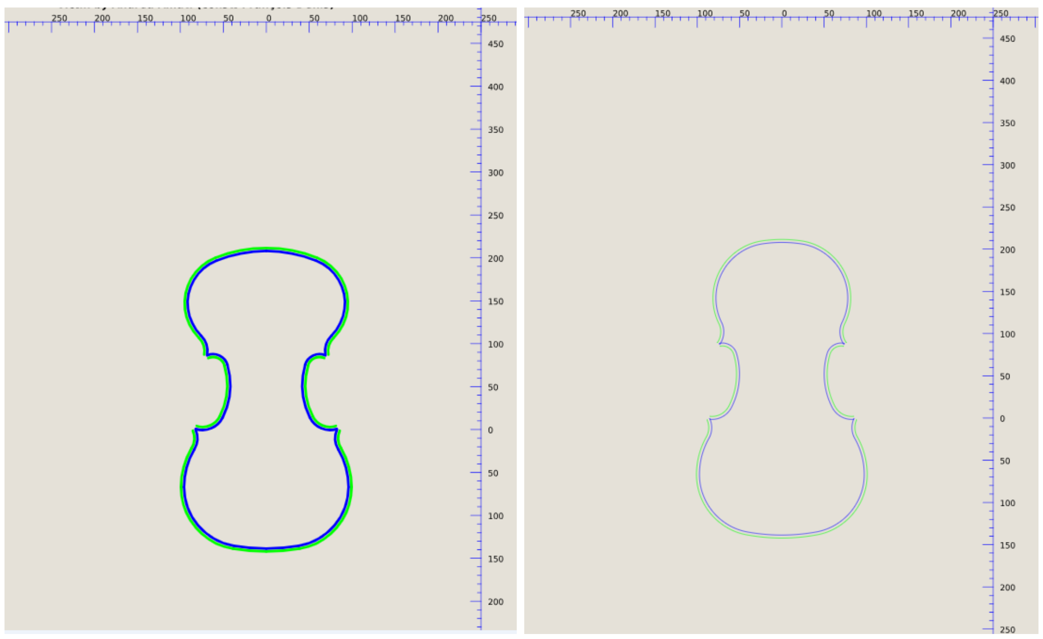


Figure 42. The difference between the underlying theme of violin outline and the original violin

CHAPTER 5

CONCLUSION

5.1 Conclusion

This study is an application of evolutionary computation. It focuses on a tiny segment of design and combines computational power with human creativity. It is a brand new attempt in this area. Based on the natural selection mechanism of the "survival of the fittest," the violin explorer simulates the evolution of violin processes. It abstracts a violin outline into a unique genotype representation. Then it uses recombination and mutation processes to change the old genotypes and create new solutions. The recombination process, in a manner, retains good violin outline characteristics. Meanwhile, the mutation process makes slight changes to the violin outline characteristics. These processes enrich the search space to a certain extent and provide users with more alternative solutions. In addition, the violin explorer adds human interactive processes to evaluate solutions, which makes it not only a simple search tool but also a creative evolutionary system. With the help of user selection activities, the violin explorer integrates personal aesthetics into the violin outline design.

In the anonymous experiment, 100 participants tested the usability of violin explorers and created a variety of violin outline design. Based on the experiment data, we found that there might have been a potential connection between the age and background of the participants and their design.

5.2 Future Work

Currently, the violin explorer provides five different models to help users design their violin. There are many different combinations of parameters that can be attempted and help users build a variety of violin designs. In addition, the violin explorer used Professor Mairson's violin design as the original violin outline. Using another violin template as the original violin outline may also affect violin design and provide a richer solution for users. Finally, the violin explorer only provides a solution for the violin body design. To make a violin, you also need to design other violin components, such as scroll, bridge, chinrest, among others. It is necessary to provide users with a solution for designing these components on the violin explorer. By using these components, users can easily form a violin and use the CNC (Computerized Numerical Control) machine to make their favorite violin.

APPENDICES

Source code(ViolinGenerator.java)

```
package org.digitalamati;
import org.apache.batik.swing.JSVGCanvas;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.jackson.JacksonConverterFactory;
import retrofit2.converter.scalars.ScalarsConverterFactory;
import javax.swing.*;
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

public class ViolinGenerator {
    /*
    Main function
     */
    public static void main(String[] args) {
        // Create a new JFrame.
        JFrame f = new JFrame("Violin Generator");
        ViolinGenerator app = new ViolinGenerator(f);
        // Add components to the frame.
        f.getContentPane().add(app.createComponents());
        // Display the frame.
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
        f.setSize(1600, 900);
        f.setVisible(true);
    }
    //the original code to draw a violin, written in AmatiML language
    protected String code = "(require AmatiML)\n" +
            "\n" +
            "(elaboration #f)\n" +
            "(mirroring #t)\n" +
            "(arcthickness 5)\n" +
            "(arccolor \"blue\")\n" +
            "(edge-tracing #t violin-overhang)\n" +
            "\n" +
            "(coded-by \"Harry Mairson\")\n" +
            "(title \"Violin by Andrea Amati (const. François Denis)\")\n" +
            "\n" +
            "; Violin by Andrea Amati\n" +
            "\n" +
            "(define (Amati)\n" +
            "\n" +
            "  ; LAYOUT OF THE AREA FRAMEWORK on which the curves are
drawn...\n" +
            "\n" +
            "(let* ((xq 208) ;; 208mm in the Amati\n" +
            " (X (label \"X\" origin))\n" +
            "(A (label \"A\" (xshift X (- (/ xq 2)))))\n" +
            " (Q (label \"Q\" (yshift X xq)))\n" +
            " (N (label \"N\" (pointfrom X Q (/ 1 4))))\n" +
            " (q (label \"q\" (xshift (at A Q) (/ (distance X N) {q}))))\n" +
            " (qp (label \"q'\" (mirror q)))\n" +
            " (O (label \"O\" (yshift Q (- (* (distance X N) (/ {O} 4))))))\n" +
            " (Z (label \"Z\" (yshift N (* (distance X N) (/ 2 3)))))\n" +
            "(P (label \"P\" (yshift X (- (* (distance X N) (/ {P} 3))))))\n" +
            "(p (label \"p\" (xshift (at A P) (/ (distance X N) 8))))\n" +
            "(pp (label \"p'\" (mirror p)))\n" +
            "(M (label \"M\" (pointfrom X P (/ 1 2))))\n" +
            "(a (label \"a\" (xshift A (/ (distance X Z) 2))))\n" +
            "(b (label \"b\" (xshift Z (- (/ (distance A a) 2)))))\n" +
            "(e (label \"e\" (xshift (at b N) (- (* (xdistance b p) (/ 3 {e}))))))\n" +
            "(c (label \"c\" (xshift (at p X) (/ (xdistance e p) {c}))))\n" +
            "(d (label \"d\" (xshift (at p X) (/ (xdistance e p) 2))))\n" +
            "(h (label \"h\" (xshift (at e Z) (- (/ (xdistance e p) 4)))))\n" +
            "(g (label \"g\" (xshift (at e Z) (- (/ (xdistance e p) 2)))))" +
            "\n" +
            "; ***** DRAWING THE OUTLINE *****\n" +
            "\n" +
            " ; THE LOWER BOUTS...\n" +
            "\n" +
            "(R1lower (circlefrom Z P))\n" +
            " (R2lower (lower-left-flank (vertical p) R1lower (distance M P)))\n" +
            " (R3lower (left-flush R2lower (distance X Z)))\n" +
            " (R4lower (lower-corner R3lower (/ (distance X N) 2) c))\n" +
            " (lower-curve (make-curve P c (list R1lower R2lower R3lower R4lower)))\n"
+
            "\n" +
            "; THE UPPER BOUTS...\n" +
            "\n" +
            " (R1upper (circlefrom N Q))\n" +
            " (R2upper (upper-left-flank (vertical q) R1upper (distance O Q)))\n" +
            " (R3upper (upper-corner R2upper (/ (distance X N) 2) g))\n" +
            " (upper-curve (make-curve Q g (list R1upper R2upper R3upper)))\n" +
            "\n" +
            "; THE MIDDLE BOUTS...\n" +
            "\n" +
            " (R1middle (circlefrom (xshift e (- (distance X Z))) e))\n" +
            " (R2middle (middle-top-corner R1middle (/ (distance N Z) 2) h))\n" +
            " (R3middle (middle-bottom-corner R1middle (/ (distance X N) 2) d))\n" +
            " (middle-curve (make-curve g c (list R2middle R1middle R3middle))))\n" +
            "\n" +
            " (list X A Q N q qp O Z P p pp M a b e c d h g \n" +
            " (horizontal N) (horizontal O) (horizontal Z)\n" +
            " (horizontal P) (horizontal Q) (horizontal X) (horizontal M)\n" +
            " (vertical p) (vertical q) (vertical b) (vertical e) \n" +
            "\n" +
            "  (line p (mirror q))\n" +
            "\n" +
            " R1lower R2lower R3lower R4lower \n" +
            " R1upper R2upper R3upper \n" +
            "R1middle R2middle R3middle \n" +
            "lower-curve middle-curve upper-curve)\n" +
            "))\n" +
            "\n" +
            "(sketch (Amati))\n" +
            "\n" +
            "(end-drawing)";
    //default value of the original violin
    final int DEFAULT_Q = 2;
    final int DEFAULT_O = 5;
    final int DEFAULT_P = 8;
    final int DEFAULT_C = 4;
    final int DEFAULT_E = 8;
    //Set size for population pool
    final int POP_SIZE = 10;
    //Set the number of models
    final int MAX_MODEL = 5;
    //Set maximum generations of each model
    final int MAX_GENERATION = 6;
    // The frame.
    protected JFrame frame;
    //create evolutionary models
    protected ES[] es = new ES[MAX_MODEL];
    //set DNA size for each evolutionary models
    protected int[] dnaSize = new int[]{5, 5, 5, 4, 5};
    //set minimum boundary for each models
    protected double[][] min = new double[][]{new double[]{2, 3, 2, 7, 8}, new
double[]{2, 3, 2, 11, 8}, new double[]{5, 3, 2, 7, 8}, new double[]{1, 4, 6, 2}, new
double[]{5, 3, 2, 11, 8}};
    //set maximum boundary for each models
    protected double[][] max = new double[][]{new double[]{4, 5, 6, 9, 10}, new
double[]{4, 5, 6, 13, 10}, new double[]{8, 5, 6, 9, 10}, new double[]{8, 5, 10, 6}, new
double[]{8, 5, 6, 13, 10}};
    //API
    protected DigitalamatiService digitalamatiService;
    //Listener for normal models
    protected Listener listener1;
    //Listener for final selection
    protected Listener listener2;
    // The svg canvas
    protected JSVGCanvas[] jsvgCanvas = new JSVGCanvas[POP_SIZE];
    //The checkboxes
    protected JCheckBox[] jCheckBoxes = new JCheckBox[POP_SIZE];
    //The jLabel
    protected JLabel jLabel = new JLabel();
    //The buttons
    protected JButton okButton;
    //Initial model to model 1
    protected int model = 1;
    //Initial generation to generation 1
    protected int generation = 1;
    //Used for Violin naming
    protected String time;
    protected ArrayList<String> resultTime = new ArrayList();
    //Used for Violin naming
    protected ArrayList<Integer> resultGeneration = new ArrayList();
    //Store the code of user selected violin
    protected ArrayList<String> resultCode = new ArrayList();
    /*
```

```
   violinGenerator
    */
   public ViolinGenerator(JFrame f) {
      frame = f;
      //Create a Retrofit object
      Retrofit retrofit = new Retrofit.Builder()
            .baseUrl("http://api.digitalamati.org/")
            .addConverterFactory(ScalarsConverterFactory.create())
            .addConverterFactory(JacksonConverterFactory.create())
            .build();
      //associate retrofit with DigitalamatiService Interface
      digitalamatiService = retrofit.create(DigitalamatiService.class);
      //five different models
      for (int i = 0; i < 5; i++) {
         es[i] = new ES(POP_SIZE, dnaSize[i], min[i], max[i]);
      }
      for (int i = 0; i < POP_SIZE; i++) {
         jsvgCanvas[i] = new JSVGCanvas();
         jCheckBoxes[i] = new JCheckBox();
         jCheckBoxes[i].addItemListener(listener2);
      }
      initModel();
   }
   /*
   initialModel method get current time to name svg image and get svg image
    */
   public void initModel() {
      getTime();
      getSvg();
   }
   /*
   Record the time when user select a violin design and use it to name the violin
design
    */
   public void getTime() {
      Date currentTime = new Date();
      SimpleDateFormat formatter = new SimpleDateFormat("yyyyMMddHHmmss");
      String dateString = formatter.format(currentTime);
      time = dateString;
      resultTime.add(time);
   }
   /*
   Get SVG image for each violin design
    */
   public void getSvg() {
      for (int i = 0; i < POP_SIZE; i++) {
            jCheckBoxes[i].setSelected(false);
            jsvgCanvas[i].setURI("empty.svg");
            getSvg(getCode(i), i);
        }
    }
    /*
    getCode method get the code for each violin. Use getDNA() method to get specific
DNA value,
    then use these value to replace original value and generate new violin design code.
    parameters:  i: the i-th violin design
    return value: - return the new violin drawing code
     */
    public String getCode(int i) {
        //get original violin drawing code
        String tempCode = code;
        if (i == 0) {
            System.out.println("model:" + model);
        }
        //generate new violin drawing code
        switch (model-1) {
            //model 1, change point q,O,c,P,e default value with new value
            case 0:
                tempCode = tempCode.replace("{q}", Double.toString(es[model-
1].getDNA()[i][0]));
                tempCode = tempCode.replace("{O}", Double.toString(es[model-
1].getDNA()[i][1]));
                tempCode = tempCode.replace("{c}", Double.toString(es[model-
1].getDNA()[i][2]));
                tempCode = tempCode.replace("{P}", Double.toString(es[model-
1].getDNA()[i][3]));
                tempCode = tempCode.replace("{e}", Double.toString(es[model-
1].getDNA()[i][4]));
                break;
            //model 2, change point q,O,c,P,e default value with new value
            case 1:
                tempCode = tempCode.replace("{q}", Double.toString(es[model-
1].getDNA()[i][0]));
                tempCode = tempCode.replace("{O}", Double.toString(es[model-
1].getDNA()[i][1]));
                tempCode = tempCode.replace("{c}", Double.toString(es[model-
1].getDNA()[i][2]));
                tempCode = tempCode.replace("{P}", Double.toString(es[model-
1].getDNA()[i][3]));
                tempCode = tempCode.replace("{e}", Double.toString(es[model-
1].getDNA()[i][4]));
                break;
```

```
          //model 3, change point q,O,c,P,e default value with new value
          case 2:
             tempCode = tempCode.replace("{q}", Double.toString(es[model-
1].getDNA()[i][0]));
             tempCode = tempCode.replace("{O}", Double.toString(es[model-
1].getDNA()[i][1]));
             tempCode = tempCode.replace("{c}", Double.toString(es[model-
1].getDNA()[i][2]));
             tempCode = tempCode.replace("{P}", Double.toString(es[model-
1].getDNA()[i][3]));
             tempCode = tempCode.replace("{e}", Double.toString(es[model-
1].getDNA()[i][4]));
             break;
          //model 4, change point q,O,c,e default value with new value
          case 3:
             tempCode = tempCode.replace("{q}", Double.toString(es[model-
1].getDNA()[i][0]));
             tempCode = tempCode.replace("{O}", Double.toString(es[model-
1].getDNA()[i][1]));
             tempCode = tempCode.replace("{c}", Double.toString(es[model-
1].getDNA()[i][3]));
             tempCode = tempCode.replace("{P}", Double.toString(DEFAULT_P));
             tempCode = tempCode.replace("{e}", Double.toString(es[model-
1].getDNA()[i][2]));
             break;
          //model 5, change point q,O,c,P,e default value with new value
          case 4:
             tempCode = tempCode.replace("{q}", Double.toString(es[model-
1].getDNA()[i][0]));
             tempCode = tempCode.replace("{O}", Double.toString(es[model-
1].getDNA()[i][1]));
             tempCode = tempCode.replace("{c}", Double.toString(es[model-
1].getDNA()[i][2]));
             tempCode = tempCode.replace("{P}", Double.toString(es[model-
1].getDNA()[i][3]));
             tempCode = tempCode.replace("{e}", Double.toString(es[model-
1].getDNA()[i][4]));
             break;
       }
       //return the changed code
       return tempCode;
    }
    /*
    Send asynchronously request to API and get the violin image
    parameters:  str: feed back from API
                 i: the i-th violin
```

```
 */
public void getSvg(String str, int i) {
   //Handle the success or failure request
   digitalamatiService.generate(str).enqueue(new Callback<Result>() {
      @Override
      public void onResponse(Call<Result> call, Response<Result> response) {
         if (response.isSuccessful()) {
            //get violin designs
            if (response.body().getErrors().equals("")) {
               String svg = response.body().getSvg();
               if (svg.indexOf("<?xml") != svg.lastIndexOf("<?xml")) {
                  getSvg(str, i);
               } else {
                  String str = svg.substring(svg.indexOf("<?"));
                  try {
                     Files.write(Paths.get("temp" + i + ".svg"), str.getBytes());
                     jsvgCanvas[i].setURI("temp" + i + ".svg");
                  } catch (IOException e) {
                     e.printStackTrace();
                  }
               }
            }
            //try again
            else {
               getSvg(str, i);
            }
         }
         //try again
         else {
            getSvg(str, i);
         }
      }
      @Override
      public void onFailure(Call<Result> call, Throwable t) {
      }
   });
}
/*
Create panel for violinGenerator,include message, image, check boxes and buttons
 */
public JComponent createComponents() {
   //create panel
   final JPanel panel = new JPanel(new BorderLayout());
   JPanel choices = new JPanel(new GridLayout(2, 5));
   //image location and checkbox location
   for (int i = 0; i < POP_SIZE; i++) {
```

```
            JPanel p = new JPanel(new BorderLayout());
            p.add("North", jCheckBoxes[i]);
            p.add("Center", jsvgCanvas[i]);
            choices.add(p);
        }
        jLabel.setText("Please select the best two violins from model " + model + "
generation " + generation);
        panel.add(jLabel, BorderLayout.NORTH);
        panel.add(choices, BorderLayout.CENTER);
        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER));
        okButton = new JButton("Generate the next generation of violin design");
        JButton skipButton = new JButton("Satisfied with the selected two design and go
to the next model");
        buttonPanel.add(okButton);
        buttonPanel.add(skipButton);
        panel.add(buttonPanel, BorderLayout.SOUTH);
        //Ok button actions
        okButton.addActionListener(e -> {
            int first = -1;
            int second = -1;
            //recode image No. of first and second selection
            for (int i = 0; i < jCheckBoxes.length; i++) {
                if (jCheckBoxes[i].isSelected()) {
                    if (first == -1)
                        first = i;
                    else
                        second = i;
                }
            }
            //must select two violin in each generation
            if (second == -1) {
                JOptionPane.showMessageDialog(null, "Please select two violin design");
            }
            boolean converge = true;
            //check if the first selection is converge
            for (double v : es[model-1].getMutationStrength()[first]) {
                if (v != 0) {
                    converge = false;
                    break;
                }
            }
            //check if the second selection is converge
            if (converge) {
                for (double v : es[model-1].getMutationStrength()[second]) {
                    if (v != 0) {
                        converge = false;
                        break;
                    }
                }
            }
            //continue evolve process until finish all models
            if (model < MAX_MODEL) {
                //continue current model, save the selected violin design in each generation
                if (!converge && generation < MAX_GENERATION) {
                    saveKid(first, second);
                    es[model-1].makeKid(first, second);
                    getSvg();
                    jLabel.setText("Please select the best two violins from model " + model +
" generation " + generation);
                }
                //go to next model
                else {
                    skip();
                }
            }
            //Finished all models,then go to the final selection process
            else {
                    getFinalViolin();
                    skipButton.setEnabled(false);
            }
        });
        //skip button action
        skipButton.addActionListener(e -> {
            //skip current model
            if (model < MAX_MODEL  ) {
                skip();
            }
            //skip button can not used in final selection
            else {
                getFinalViolin();
                skipButton.setEnabled(false);
            }
        });
        return panel;
    }
    /*
    Save current selections and skip current model and jump to next model
     */
    private void skip() {
        int first = -1;
        int second = -1;
        //recode image No. of selection
        for (int i = 0; i < jCheckBoxes.length; i++) {
            if (jCheckBoxes[i].isSelected()) {
                if (first == -1)
                    first = i;
                else
                    second = i;
            }
        }
        //save the code and image of current selection
        resultGeneration.add(generation);
        resultCode.add(getCode(first));
        resultCode.add(getCode(second));
        saveKid(first, second);
        generation = 1;
        model++;
        initModel();
        jLabel.setText("Please select the best two violins from model " + model + "
generation " + generation);
    }
    /*
    Let users select the final design after they go through all 5 models
     */
    private void getFinalViolin() {
        int first = -1;
        int second = -1;
        //recode image No. of selection
        for (int i = 0; i < jCheckBoxes.length; i++) {
            if (jCheckBoxes[i].isSelected()) {
                if (first == -1)
                    first = i;
                else
                    second = i;
            }
        }
        //save the code and image of current selection
        resultGeneration.add(generation);
        resultCode.add(getCode(first));
        resultCode.add(getCode(second));
        saveKid(first, second);
        //display all 10 selections in previous models
        for (int i = 0; i < resultTime.size(); i++) {
            //first selection of each model
            jCheckBoxes[i * 2].setSelected(false);
            jCheckBoxes[i * 2].removeItemListener(listener2);
            jCheckBoxes[i * 2].addItemListener(listener1);
            jsvgCanvas[i * 2].setURI("empty.svg");
            jsvgCanvas[i * 2].setURI("kid-" + resultTime.get(i) + "-" +
resultGeneration.get(i) + "-" + 0 + ".svg");
            //second selection of each model
            jCheckBoxes[i * 2 + 1].setSelected(false);
            jCheckBoxes[i * 2 + 1].removeItemListener(listener2);
            jCheckBoxes[i * 2 + 1].addItemListener(listener1);
            jsvgCanvas[i * 2 + 1].setURI("empty.svg");
            jsvgCanvas[i * 2 + 1].setURI("kid-" + resultTime.get(i) + "-" +
resultGeneration.get(i) + "-" + 1 + ".svg");
        }
        jLabel.setText("Please select the final violin");
        okButton.removeActionListener(okButton.getActionListeners()[0]);
        okButton.addActionListener(e -> {
            getTime();
            //Display user's final selection and save the image and code of that violin
design
            for (int i = 0; i < jCheckBoxes.length; i++) {
                if (jCheckBoxes[i].isSelected()) {
                    try {
                        Files.copy(Paths.get("kid-" + resultTime.get(i / 2) + "-" +
resultGeneration.get(i / 2) + "-" + i % 2 + ".svg"), Paths.get("result-" + time + ".svg"),
StandardCopyOption.REPLACE_EXISTING);
                        Files.write(Paths.get("result-" + time + ".txt"),
resultCode.get(i).getBytes());
                        frame.setVisible(false);
                        JFrame f = new JFrame("Result");

                        JSVGCanvas canvas = new JSVGCanvas();
                        canvas.setURI("result-" + time + ".svg");
                        f.add(canvas);

                        // Display the frame.
                        f.addWindowListener(new WindowAdapter() {
                            public void windowClosing(WindowEvent e) {
                                System.exit(0);
                            }
                        });
                        f.setSize(1600, 900);
                        f.setVisible(true);
                    } catch (IOException e1) {
                        e1.printStackTrace();
                    }
                    break;
                }
            }
        });
```

```
    }
    /*
    Help user to save violin designs they selected
    parameters:  first: first selection in each generation
                 second: second selection in each generation
     */
    private void saveKid(int first, int second) {
       //save users' selection in each generation and models
       try {
          Files.copy(Paths.get("temp" + first + ".svg"), Paths.get("kid-" + time + "-" +
generation + "-0.svg"), StandardCopyOption.REPLACE_EXISTING);
          Files.copy(Paths.get("temp" + second + ".svg"), Paths.get("kid-" + time + "-" +
generation + "-1.svg"), StandardCopyOption.REPLACE_EXISTING);
          generation++;
       } catch (IOException e1) {
          e1.printStackTrace();
       }
    }
    public class Listener implements ItemListener {
       private int maxSelection;
       private int selectionCounter = 0; //initial counter to 0
       public Listener(int maxSelection) {
          this.maxSelection = maxSelection;
       }
       @Override
       public void itemStateChanged(ItemEvent e) {
          JCheckBox source = (JCheckBox) e.getSource();
          if (source.isSelected()) {
             selectionCounter++;
             // check for max selections:
             if (selectionCounter == maxSelection)
                for (JCheckBox box : jCheckBoxes)
                   if (!box.isSelected())
                      box.setEnabled(false);
          } else {
             selectionCounter--;
             // check for less than max selections:
             if (selectionCounter < maxSelection)
                for (JCheckBox box : jCheckBoxes)
                   box.setEnabled(true);
          }
       }
    }
}
```

Source code(ES.java)

```
package org.digitalamati;
import java.util.Arrays;
/*
Use evolution strategies algorithm to evolve violin outline
*/
public class ES {
   private int popSize;
   private int dnaSize;
   private double[][] DNA;
   private double[][] mutationStrength;
   private double[] min;
   private double[] max;

   /*
   Use evolution strategies algorithm to evolve violin outline
   parameters:  popSize: the size of population pool(set to 10)
             dnaSize: the number of elements for each violin DNA
             min: the minimum value of each DNA element
             max: the maximum value of each DNA element
   */
   public ES(int popSize, int dnaSize, double[] min, double[] max) {
      this.popSize = popSize;
      this.dnaSize = dnaSize;
      this.min = min;
      this.max = max;

      //Randomly generate DNA and mutation strengh to create original population
      DNA = new double[popSize][dnaSize];
      mutationStrength = new double[popSize][dnaSize];
      for (int i = 0; i < popSize; i++) {
         for (int j = 0; j < dnaSize; j++) {
            DNA[i][j] = min[j] + Math.random() * (max[j] - min[j]);
            mutationStrength[i][j] = Math.random();
            DNA[i][j] = Math.max(min[j], DNA[i][j]);
            DNA[i][j] = Math.min(max[j], DNA[i][j]);
            DNA[i][j] = Double.parseDouble(String.format("%.1f", DNA[i][j]));
            mutationStrength[i][j] = Double.parseDouble(String.format("%.2f",
mutationStrength[i][j]));
         }
      }
      System.out.println("DNA:" + Arrays.deepToString(DNA));
```

```
        System.out.println("mutationStrength:" +
Arrays.deepToString(mutationStrength));
    }
    /*
    return DNA of each violin
    return value:
    -the violin's DNA
     */
    public double[][] getDNA() {
        return DNA;
    }
    /*
    return mutation strength of each violin
    return value:
    -the violin's mutation strength
     */
    public double[][] getMutationStrength() {
        return mutationStrength;
    }

    /*
    simulate the evolution prosesses, use crossover and mutate to create new generation
population
    parameters:  first: the first selection of users
                        second: the second selection of users
     */
    public void makeKid(int first, int second) {
        //Create arrays to store offspring's DNA and mutation strength
        double[][] kidsDNA = new double[popSize][dnaSize];
        double[][] kidsMutationStrength = new double[popSize][dnaSize];
        //The recombination and mutation processes
        for (int i = 0; i < popSize; i++) {
            //bp determine the position to break a DNA into two pieces
            int bp = (int) (Math.random() * dnaSize);
            //rnd determine which parent provide the first piece DNA to kid, the other
provide the second piece
            double rnd = Math.random();
            int tempFirst = rnd > 0.5 ? first : second;
            int tempSecond = rnd > 0.5 ? second : first;
            //recombination process
            System.arraycopy(DNA[tempFirst], 0, kidsDNA[i], 0, bp);
            System.arraycopy(DNA[tempSecond], bp, kidsDNA[i], bp, dnaSize - bp);
            System.arraycopy(mutationStrength[tempFirst], 0, kidsMutationStrength[i], 0,
bp);
            System.arraycopy(mutationStrength[tempSecond], bp,
kidsMutationStrength[i], bp, dnaSize - bp);
```

```
            //mutation process
            for (int j = 0; j < dnaSize; j++) {
                //mutation strength decrease in each generation to convergence faster
                kidsMutationStrength[i][j] = mutationStrength[i][j] * 0.9;
                kidsMutationStrength[i][j] = kidsMutationStrength[i][j] < 0.05 ? 0 :
kidsMutationStrength[i][j];
                //use mutation strength to slightly change kid's DNA
                int num = Math.random() > 0.5 ? 1 : -1;
                kidsDNA[i][j] += num * kidsMutationStrength[i][j] * Math.random();
                //to make sure each element of DNA within its scope
                kidsDNA[i][j] = Math.max(min[j], kidsDNA[i][j]);
                kidsDNA[i][j] = Math.min(max[j], kidsDNA[i][j]);
                kidsDNA[i][j] = Double.parseDouble(String.format("%.1f",
kidsDNA[i][j]));
                kidsMutationStrength[i][j] = Double.parseDouble(String.format("%.2f",
kidsMutationStrength[i][j]));
            }
        }
        //put kids' DNA and mutation strength to the population pool
        DNA = kidsDNA;
        mutationStrength = kidsMutationStrength;
        System.out.println("kid DNA:" + Arrays.deepToString(DNA));
        System.out.println("kid mutationStrength:" +
Arrays.deepToString(mutationStrength));
    }
}
```

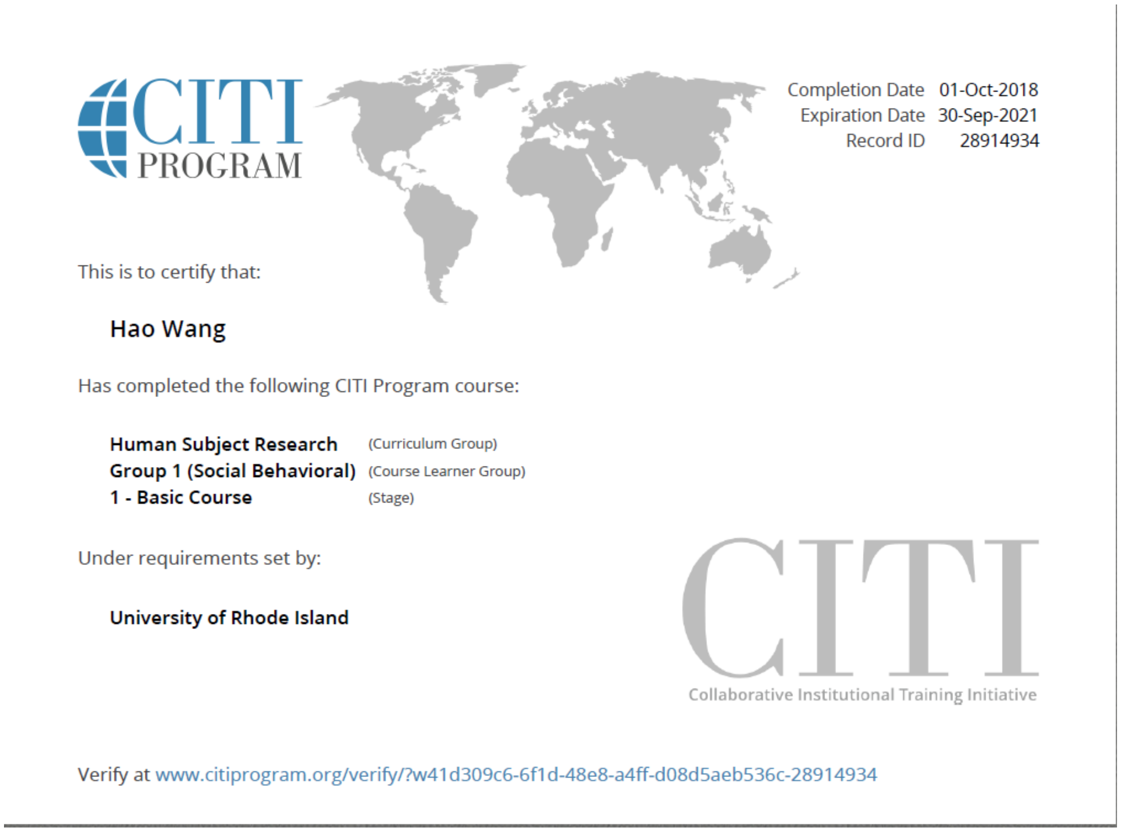
Source code(result.java)

```
package org.digitalamati;
/*
Entity class to receive feedback
 */
public class Result {
    private String errors;
    private String svg;

    public String getErrors() {
        return errors;
    }

    public void setErrors(String errors) {
        this.errors = errors;
    }

    public String getSvg() {
        return svg;
    }

    public void setSvg(String svg) {
        this.svg = svg;
    }
}
```

Source code(DigitalamatiService.java)

```
package org.digitalamati;

import retrofit2.Call;
import retrofit2.http.*;
/*
Used to build a API for violin explorer
 */
public interface DigitalamatiService {
   @POST("/")
   Call<Result> generate(@Body String body);
}
```

CitiCompletionReport



Completion Date 01-Oct-2018
Expiration Date 30-Sep-2021
Record ID 28914934

This is to certify that:

Hao Wang

Has completed the following CITI Program course:

Human Subject Research	(Curriculum Group)
Group 1 (Social Behavioral)	(Course Learner Group)
1 - Basic Course	(Stage)

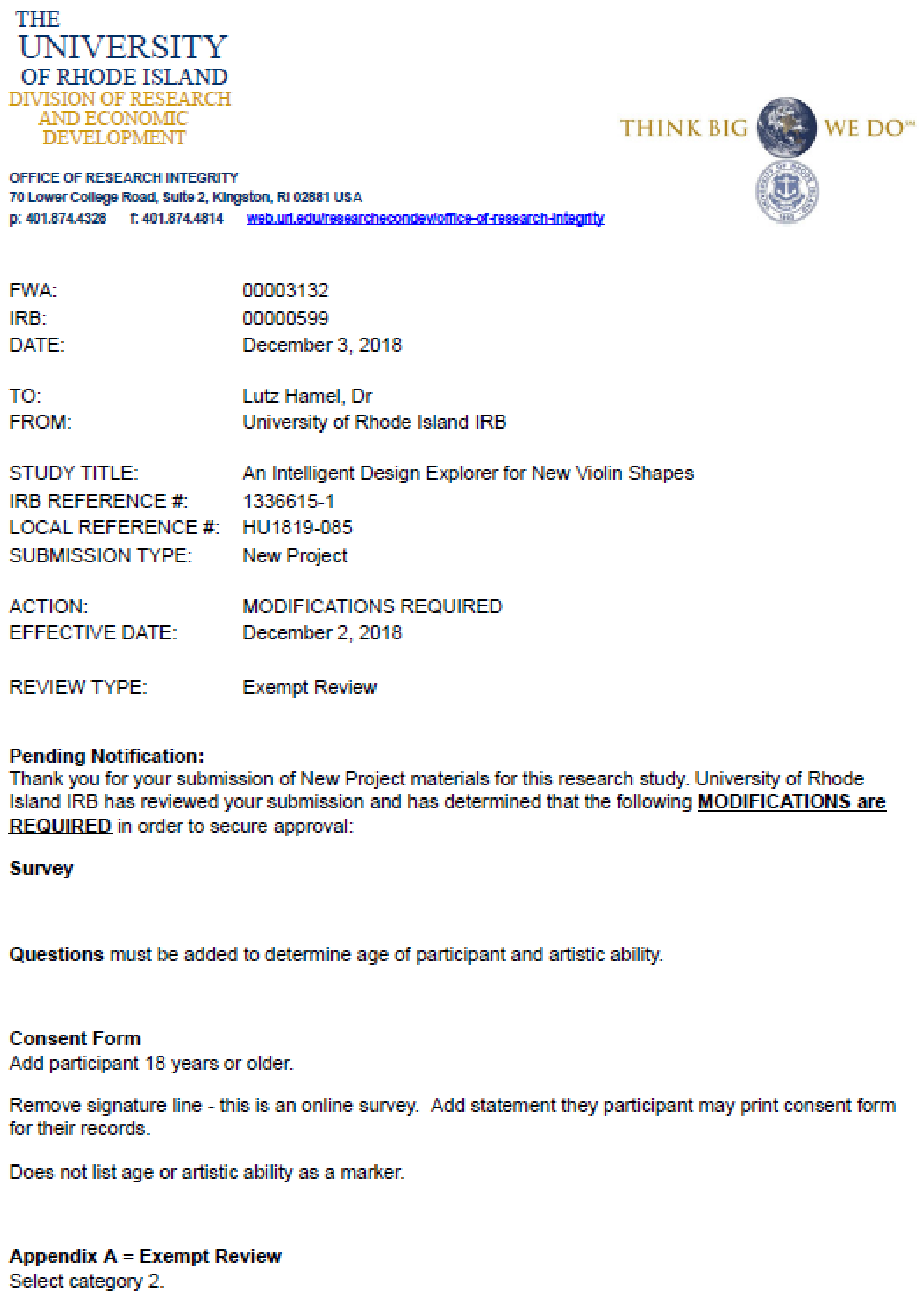
Under requirements set by:

University of Rhode Island

Collaborative Institutional Training Initiative

Verify at www.citiprogram.org/verify/?w41d309c6-6f1d-48e8-a4ff-d08d5aeb536c-28914934

IRBNetDocument



THE UNIVERSITY OF RHODE ISLAND
DIVISION OF RESEARCH AND ECONOMIC DEVELOPMENT

THINK BIG WE DO℠

OFFICE OF RESEARCH INTEGRITY
70 Lower College Road, Suite 2, Kingston, RI 02881 USA
p: 401.874.4328 f: 401.874.4814 web.uri.edu/researchecondev/office-of-research-integrity

FWA:	00003132
IRB:	00000599
DATE:	December 3, 2018
TO:	Lutz Hamel, Dr
FROM:	University of Rhode Island IRB
STUDY TITLE:	An Intelligent Design Explorer for New Violin Shapes
IRB REFERENCE #:	1336615-1
LOCAL REFERENCE #:	HU1819-085
SUBMISSION TYPE:	New Project
ACTION:	MODIFICATIONS REQUIRED
EFFECTIVE DATE:	December 2, 2018
REVIEW TYPE:	Exempt Review

Pending Notification:
Thank you for your submission of New Project materials for this research study. University of Rhode Island IRB has reviewed your submission and has determined that the following **MODIFICATIONS are REQUIRED** in order to secure approval:

Survey

Questions must be added to determine age of participant and artistic ability.

Consent Form
Add participant 18 years or older.

Remove signature line - this is an online survey. Add statement they participant may print consent form for their records.

Does not list age or artistic ability as a marker.

Appendix A = Exempt Review
Select category 2.

IRB Application
Section 1. End date should be at least one year from now.

Section 11b. Also check "Internet or e-mail collection"

Section 14.b. Select URI students.

Section 14 d. You are selecting on age.

Section 14e. Students are vulnerable to coercion so this should be "yes" and provide a brief explanation on how this will be minimized.

Section 17. Must be answered.

Section 18. Also select "Informed Consent - Form"

Section 23. Risk is minimal.

Include with your submission a brief memo listing the modifications made and a copy of each required document with the changes either highlighted, underscored, or in bold print.

To submit your revisions:

1. *Select your protocol [1336615-1].*
2. *On the left hand side of the page, select PROJECT HISTORY.*
3. *At the bottom of the page, select CREATE NEW PACKAGE.*
4. *Upload any modified or new documents.*
5. *Upload a brief memo describing your changes.*
6. *On the left hand side of the page, select SIGN THIS PACKAGE.*
7. *On the left hand side, select SUBMIT.*

Research activities in accordance with this submission may not begin until this office has received a response to these conditions and issued final approval.

This submission has received Exempt Review based on the applicable federal regulation.

If you have any general questions, please contact us by email at researchintegrity@etal.uri.edu. For study related questions, please contact us via **project mail through IRBNet**. Please include your study title and reference number in all correspondence with this office.

[illegible]

Matthew Delmonico, Ph.D., MPH
IRB Chair

LIST OF REFERENCE

[1]Lunenfeld, P. (2003). The design cluster. Design research: Methods and perspectives, 10-15.

[2] "Design," n.d. https://en.wikipedia.org/wiki/Design

[3] National Association Of Schools Of Art And Design. (2018) *National Association of Schools of Design Handbook.* Retrieved from https://nasad.arts-accredit.org/wp-content/uploads/sites/3/2018/12/AD-2018-19-Handbook-FINAL-12-17-2018.pdf

[4] algorithmic art assembly. https://aaassembly.org/

[5] Frazer, J. (1995). An evolutionary architecture.

[6] Soddu, C. (2001). Generative Natural Flux. In *GA 2001* (pp. 9-22). Generative design Lab.

[7] R. M. Friedberg, " A learning machine: Part I," IBM J., vol. 2, no. 1, pp. 2–13, Jan. 1958.

[8] R. M. Friedberg, B. Dunham, and J. H. North, "A learning machine: Part II," IBM J., vol. 3, no. 7, pp. 282–287, July 1959.

[9] G. E. P. Box, "Evolutionary operation: A method for increasing industrial productivity," Appl. Statistics, vol. VI, no. 2, pp. 81–101, 1957.

[10] Bentley, P. (1999). *Evolutionary design by computers*. Morgan Kaufmann.

[11] Harry Mairson. http://www.digitalamati.org/

[12] Bentley, P. (2002). *Creative evolutionary systems.* Morgan Kaufmann.

[13] Yao, X. (1999). *Evolutionary computation: Theory and applications*. World scientific.

[14] Bentley, P. (1999). An introduction to evolutionary design by computers. *Evolutionary design by computers*, 1-73.

[15] Holland, J. H. (1992). Genetic algorithms. *Scientific american*, *267*(1), 66-73.

[16] Fogel, D. B. (1993). Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and systems*, *24*(1), 27-36.

[17] Rechenberg, I. (1989). Evolution strategy: Nature's way of optimization. In *Optimization: Methods and applications, possibilities and limitations* (pp. 106-126). Springer, Berlin, Heidelberg.

[18] Koza, J. R., & Koza, J. R. (1992). *Genetic programming: on the programming of computers by means of natural selection*(Vol. 1). MIT press.

[19] Back, T., Hammel, U., & Schwefel, H. P. (1997). Evolutionary computation: Comments on the history and current state. IEEE transactions on Evolutionary Computation, 1(1), 3-17.

[20] Frazer, J. (2002). Creative design and the generative evolutionary paradigm. In *Creative evolutionary systems* (pp. 253-274). Morgan Kaufmann.

[21] Moroni, A., Manzolli, J., Von Zuben, F., & Gudwin, R. (2002). Vox Populi: Evolutionary computation for music evolution. In *Creative evolutionary systems* (pp. 205-221). Morgan Kaufmann.

[22]Rooke, S. (2002). Eons of genetically evolved algorithmic images. In Creative evolutionary systems (pp. 339-365). Morgan Kaufmann.

[23] Retrofit 2[online]. available: https://github.com/square/retrofit

[24] Apache™ Batik SVG Toolkit[online]. available: https://xmlgraphics.apache.org/batik/.2016

BIBLIOGRAPHY

Frazer, J. (1995). An evolutionary architecture.

Yu, X., & Gen, M. (2010). *Introduction to evolutionary algorithms*. Springer Science & Business Media.

Fogel, D. B. (1998). *Evolutionary computation: the fossil record*. Wiley-IEEE Press.

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008, June). Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)* (pp. 3381-3387). IEEE.

Gabora, L. (2002). The beer can theory of creativity. In *Creative evolutionary systems* (pp. 147-161). Morgan Kaufmann.

Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Miranda, E. R. (2007). *Evolutionary computer music* (p. xiv259). J. Al Biles (Ed.). London: Springer.

Bentley, P. J., & Corne, D. W. (2002). An introduction to creative evolutionary systems. In *Creative evolutionary systems* (pp. 1-75). Morgan Kaufmann.

Bäck, T., Hammel, U., & Schwefel, H. P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE transactions on Evolutionary Computation*, *1*(1), 3-17.

Soddu, C., & Colabella, E. Artificial intelligence and architectural design. In *CAAD Futures* (Vol. 95).