Genetic Algorithms and Shape Grammars

Technical report

Author	Manuela Ruiz Montiel
Date	October 18, 2011
Version	1.1

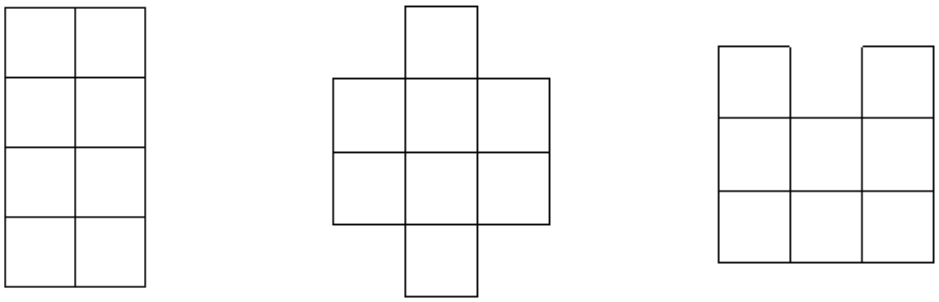
Contents

1. Introduction

Shape grammars [1] are a powerful tool to generate different designs from a fixed set of rules. In addition, we may want the synthesized solutions to satisfy some criteria or constraints. A particular way to achieve it is by using evolutionary optimization techniques.

We have used a genetic algorithm [2] in order to evolve *rule sequences,* as in many works that can be found in the literature. One of the first examples is the one of Gero et al [4]. Each sequence represents a shape, thus allowing us to measure the shape performance according to certain design criteria, i.e., objective functions.

The aim of this technical report is to illustrate the integration between a genetic algorithm and a shape grammar, using a simple design problem: generate figures of 8 tiles vertically symmetric, for example:



We will use a simple, one-rule shape grammar that produces all the possible shapes with 8 tiles:

Rule: 

Axiom: 

Starting from the axiom, we apply seven rules so as to produce 8-tile shapes. If we apply the rules randomly, obviously there will be no guarantee for the produced shapes to be symmetric.

2. Genetic algorithm

In this section we explain the genetic algorithm that has been integrated with the shape grammar.

The underlying motivation of genetic algorithms is to model the process of natural evolution. A population of individuals evolves generation by generation by means of three operators: selection, crossover and mutation.

The individuals that are going to evolve are sequences of seven rules, applied starting from the axiom. Each sequence corresponds to a certain shape. The way of obtaining the *phenotype* (the real shape) from the *genotype* (binary string that is actually evolved) is detailed in section 3.

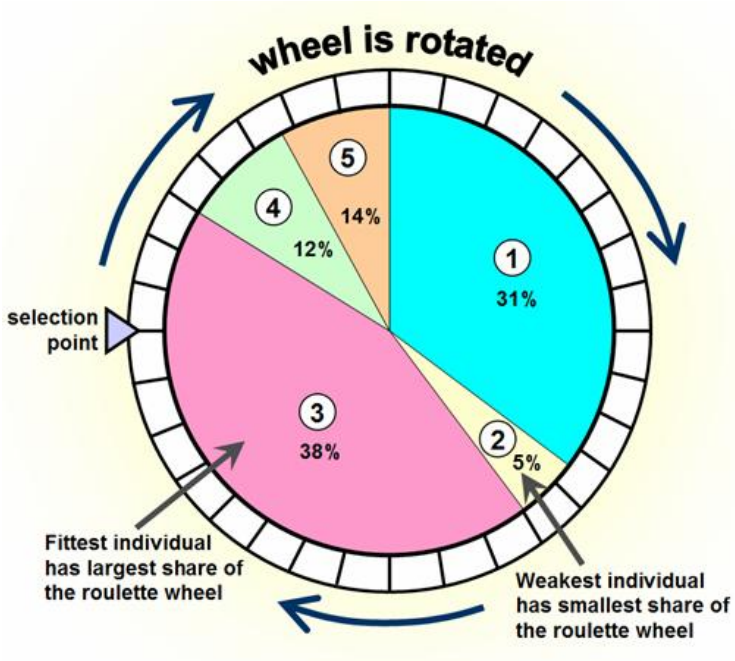
The **selection** operator has been implemented according to a fitness function, in the sense that those individuals that better perform according to our objective will we more likely to reproduce. In this case, the fitness function measures the symmetry of the shapes produced by the rule sequences, which is the objective to be maximized.

The chosen selection method for determining the set of individuals that are going to reproduce (that is, the *mating pool*) is the so-called *roulette-wheel* selection. In this roulette, each individual has a slot sized in proportion to its fitness value. With a spin of the roulette, one individual will be selected and pushed into the mating pool. For example:

No.	Chromosome	$Value_{10}$	*X*	Fitness *f(x)*	% of Total
1	0001101011	107	1.05	6.82	31
2	1111011000	984	9.62	1.11	5
3	0100000101	261	2.55	8.48	38
4	1110100000	928	9.07	2.57	12
5	1110001011	907	8.87	3.08	14
Totals				22.05	100

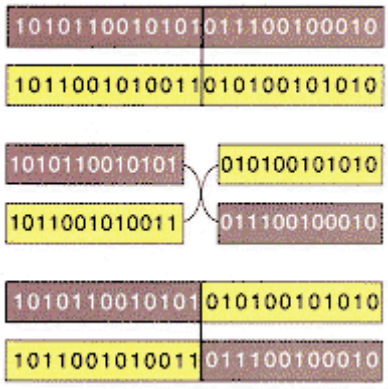
Example population of 5 for: $f(x) = -\frac{1}{4}x^2 + 2x + 5$

Extracted from http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/



Extracted from http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/

Once the mating pool is populated by consecutive roulette spins, its members are mated at random and, by means of the **crossover** operator, two new individuals will be generated from each pair. This operator works with the string genotypes, choosing a random position and swapping the contents of the initial strings from this position. The crossover process ends when the mating pool is empty.



Crossover process
Extracted from http://www.fortunecity.com/emachines/e11/86/algo.html

These new individuals are then mutated with some probability. **Mutation** consists of changing the value of a random position inside the genotype string.

The pseudocode for the algorithm is given below:

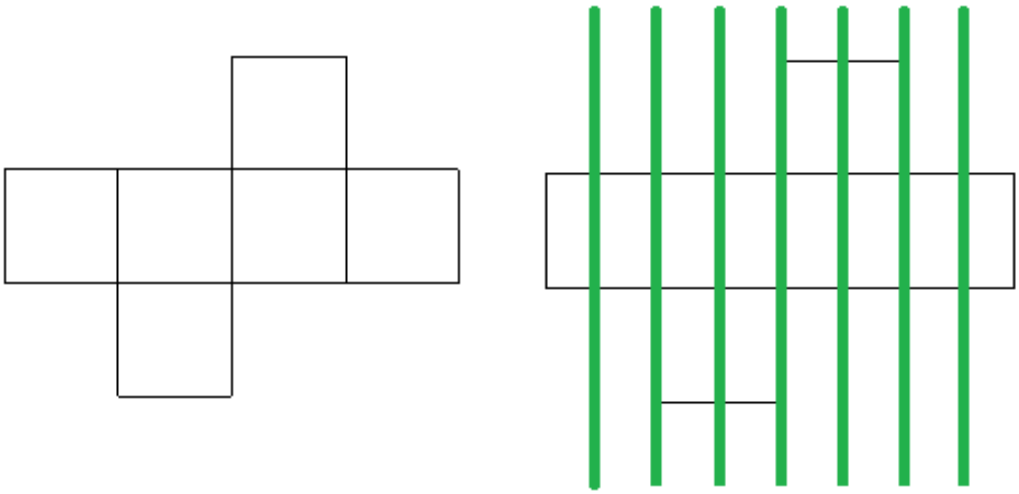
```
g = 0
initialize P(g) randomly
while (g < number of generations) do
     select P(g+1)
     crossover P(g+1)
     mutate P(g+1)
     g = g+1
end
```

We have to choose some parameters of the algorithm:

- Population size: number of individuals that populate each generation
- Number of generations: number of cycles of the algorithm
- Mutation rate: probability for an individual to mutate.

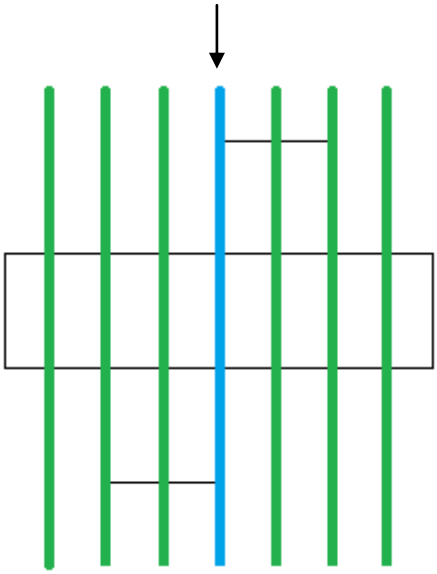
In addition, the fitness function has been considered in two different flavours. Each experiment has been made twice:

1. With a fitness function that returns 1 if the shape is symmetric and 0 otherwise
2. With a more relaxed fitness function that counts the number of tiles that are paired-up with respect to a central vertical axis. The maximum value of this function is 8. The central vertical axis is chosen among the possible ones, as in the next figure:



A shape (left) and its set of possible axis (right)

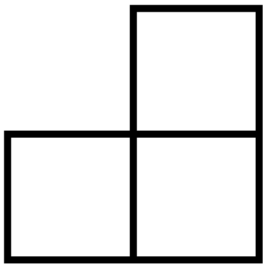
For every possible axis, we calculate the number of tiles that exist at its right and at its left side, and make the absolute difference. The axis that has a lower difference is the chosen one. In case of draw the axis chosen is the one with a lower x value:



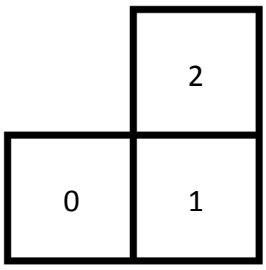
The chosen axis

3. Genotype

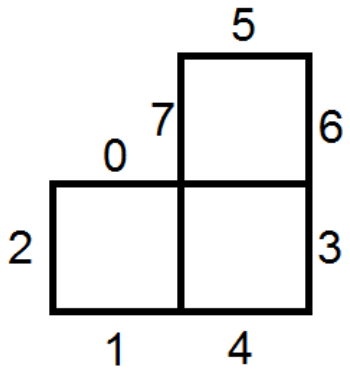
The genotype is a binary string of seven *codons*, one for each rule that is applied. Each codon is a binary number of four digits, so the total length of the genotype is of 28 bits. Each codon identifies the rule to be applied at each step. Suppose that at certain step we have the next shape:



The tiles are ordered with respect to the *(x,y)* coordinates of their central point, lexicographically. So the ordering of the tiles of the previous shape is the following:

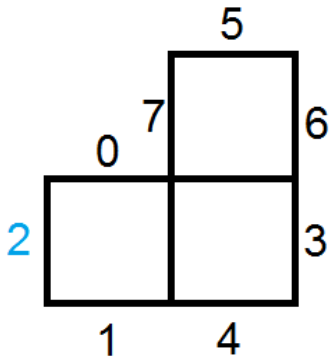


The rule to apply is to be chosen from the set of possible rules. Starting from the first tile, the set of possible rules is made up searching for empty neighbours, starting from the top and then in clockwise order for each tile:



The rule 0 would be the one that takes the first tile (the one numbered with a 0) as its left part and adds a neighbour at the top, the rule 1 would be the one that adds it at the bottom, and so on.

If the next codon is 0010, that is, it has a decimal value of 2, then the rule to apply is the one that adds a left neighbour to the first tile:



If the codon to apply to this shape is greater than 7, then we use (codon mod 8), as 8 is the total number of possible rules. For example, if the next codon is 1010, that is, it has a decimal value of 10, then the rule to apply is the same as in the previous example. This way, we assure that every genotype is a valid one.

The length of codons is of four digits because the maximum number of rules that can be applied at the one given step is 16.

4. Experiments

We have developed 24 distinct experiments, one for every combination of the following parameters of the genetic algorithm:

- Binary fitness / Not binary fitness
- Number of generations: 100 – 200
- Population size: 50 – 100
- Mutation rate: 0.02 – 0.1 – 0.2

In table 1 the data of all the experiments is gathered.

#Experiment	Binary Fitness?	#Generations	Population size	Mutation rate
0	No	100	50	0.02
1	No	100	50	0.1
2	No	100	50	0.2
3	No	100	100	0.02
4	No	100	100	0.1
5	No	100	100	0.2
6	No	200	50	0.02
7	No	200	50	0.1
8	No	200	50	0.2
9	No	200	100	0.02
10	No	200	100	0.1
11	No	200	100	0.2
12	Yes	100	50	0.02
13	Yes	100	50	0.1
14	Yes	100	50	0.2
15	Yes	100	100	0.02
16	Yes	100	100	0.1
17	Yes	100	100	0.2
18	Yes	200	50	0.02
19	Yes	200	50	0.1
20	Yes	200	50	0.2
21	Yes	200	100	0.02
22	Yes	200	100	0.1
23	Yes	200	100	0.2

Table 1: Experiment data

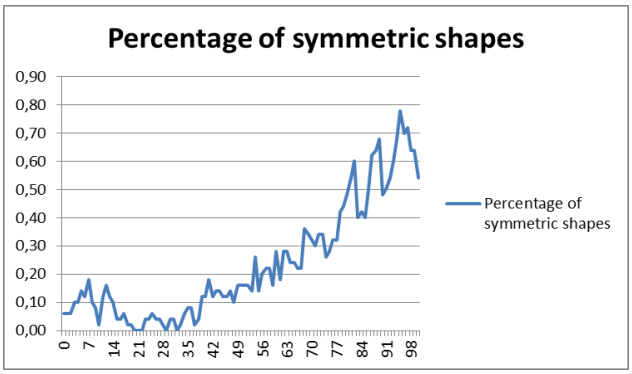
Now, we provide graphic information of:

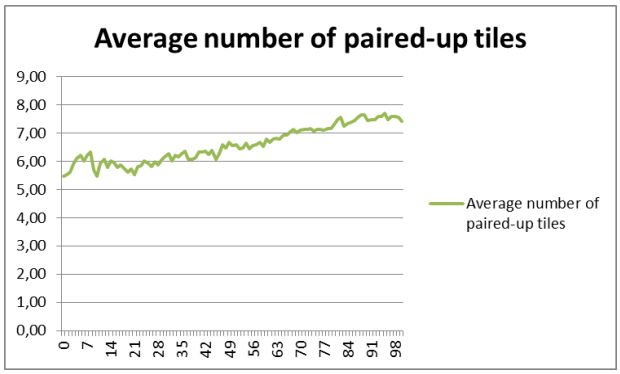
1) The percentage of symmetric shapes for every generation of every experiment
2) The average number of paired-up tiles for every generation of every experiment

Set of experiments 0-1-2

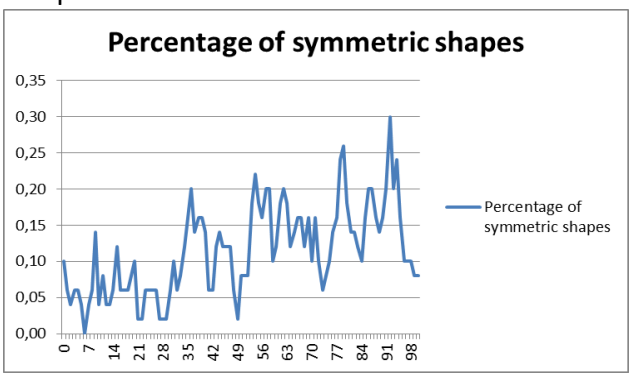
- Fitness: number of paired-up tiles
- Number of generations: 100
- Population size: 50

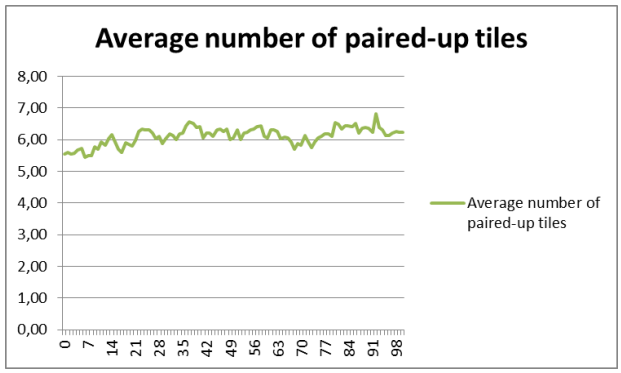
Experiment 0 – Mutation rate: 0.02



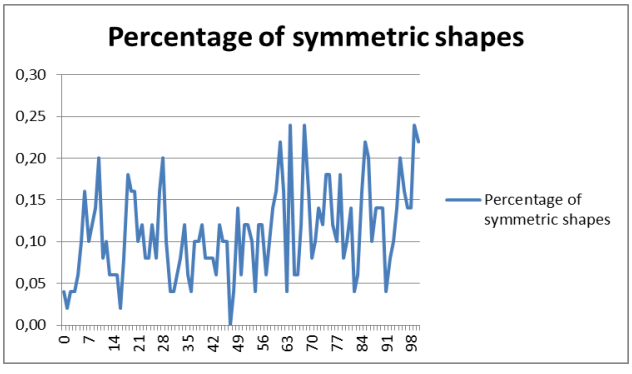


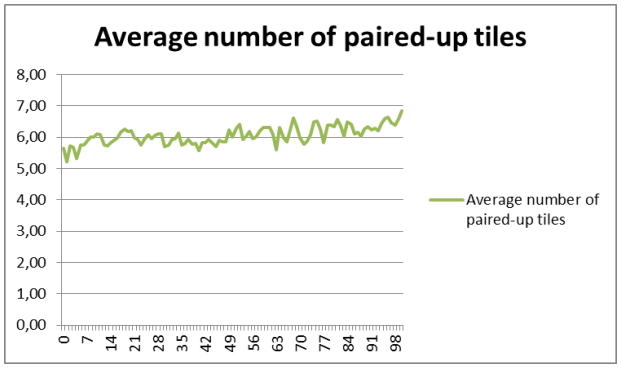
Experiment 1– Mutation rate: 0.1





Experiment 2– Mutation rate: 0.2

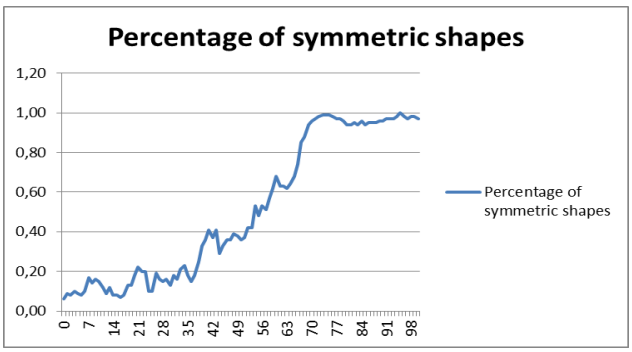


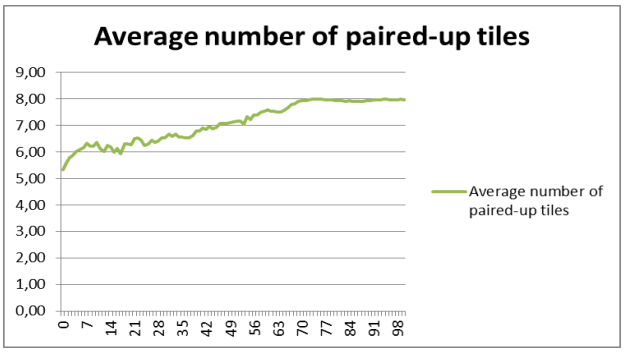


Set of experiments 3-4-5

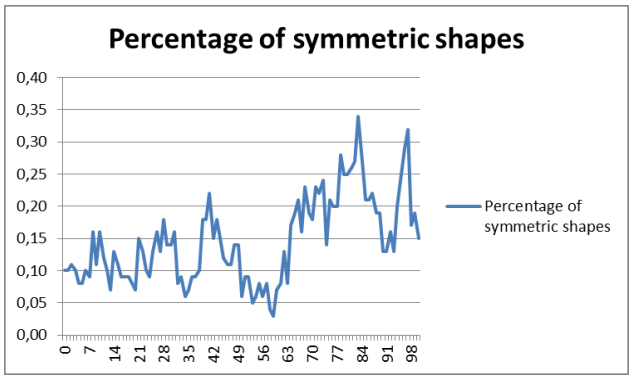
- Fitness: number of paired-up tiles
- Number of generations: 100
- Population size: 100

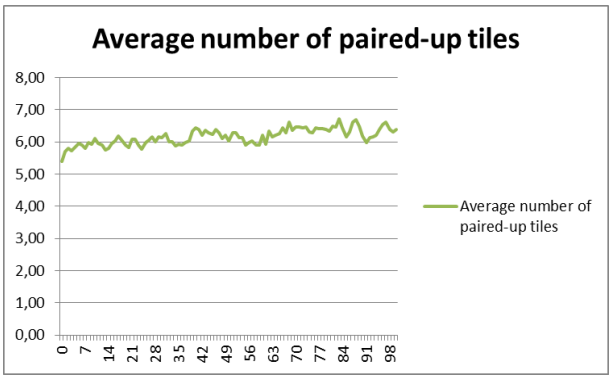
Experiment 3– Mutation rate: 0.02



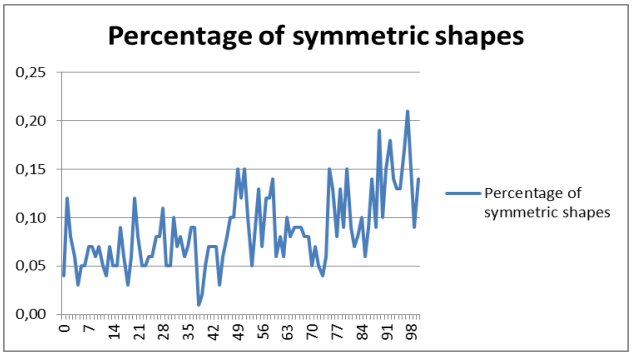


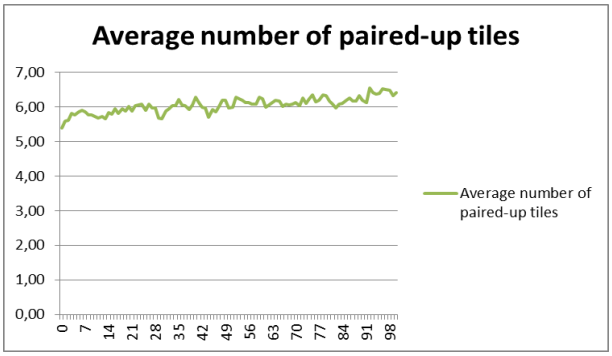
Experiment 4– Mutation rate: 0.1





Experiment 5– Mutation rate: 0.2

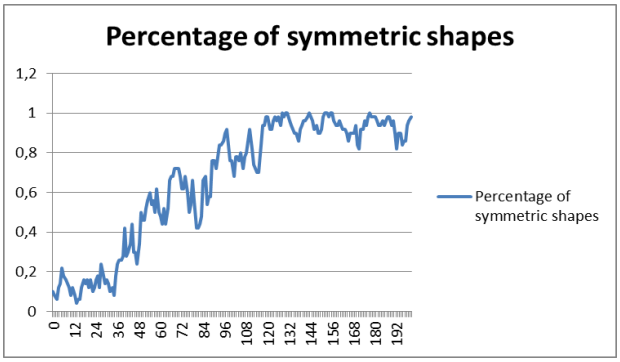


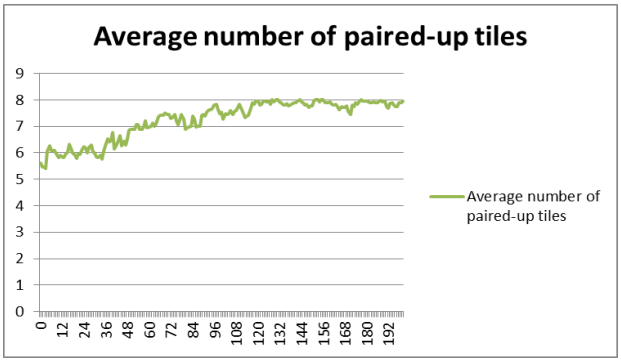


Set of experiments 6-7-8

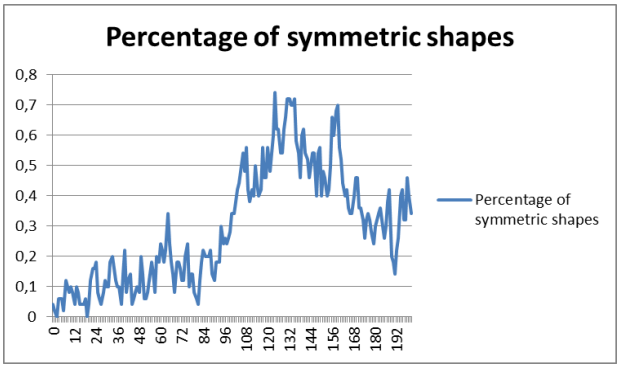
- Fitness: number of paired-up tiles
- Number of generations: 200
- Population size: 50

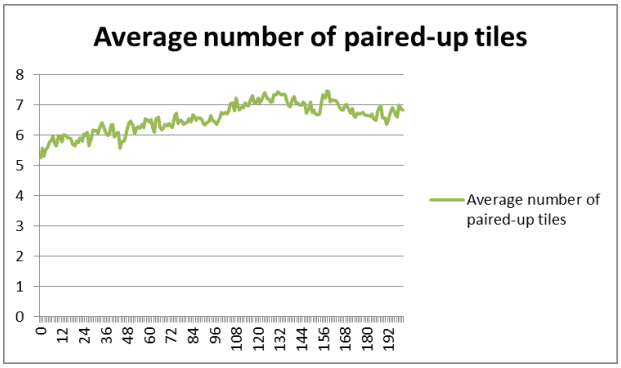
Experiment 6– Mutation rate: 0.02



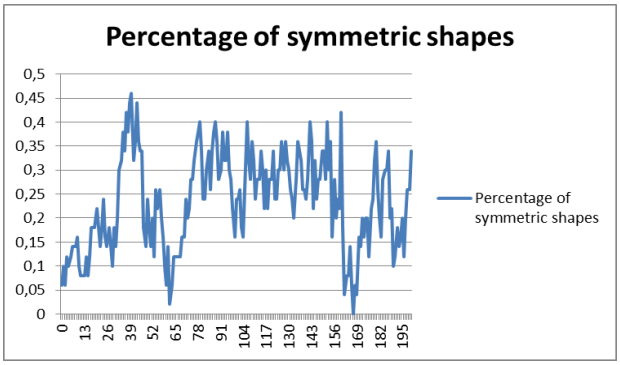


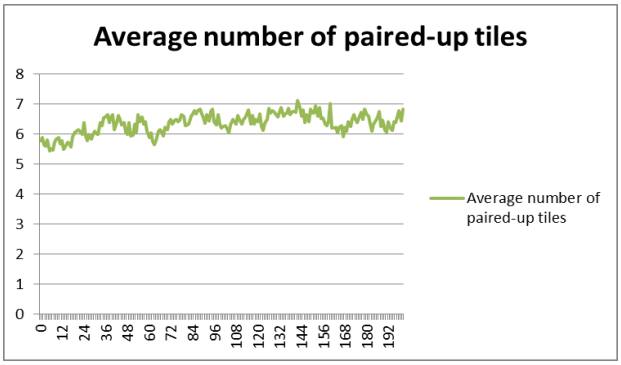
Experiment 7– Mutation rate: 0.1





Experiment 8– Mutation rate: 0.2

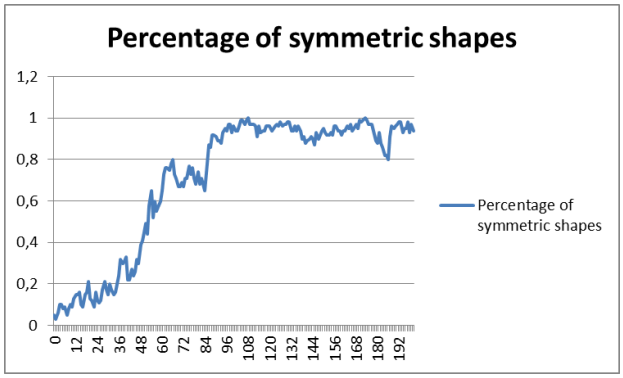


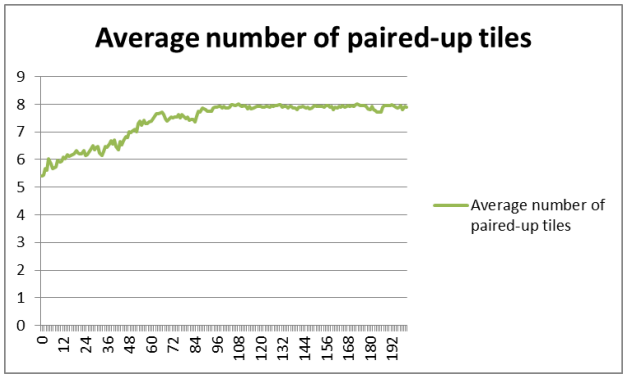


Set of experiments 9-10-11

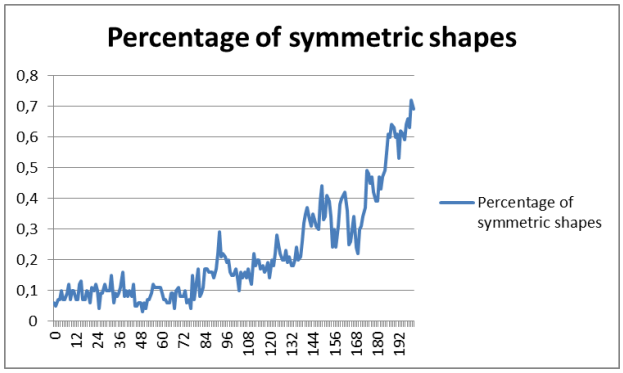
- Fitness: number of paired-up tiles
- Number of generations: 200
- Population size: 100

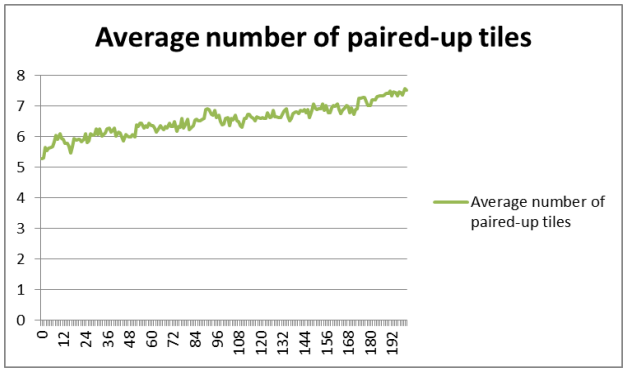
Experiment 9– Mutation rate: 0.02



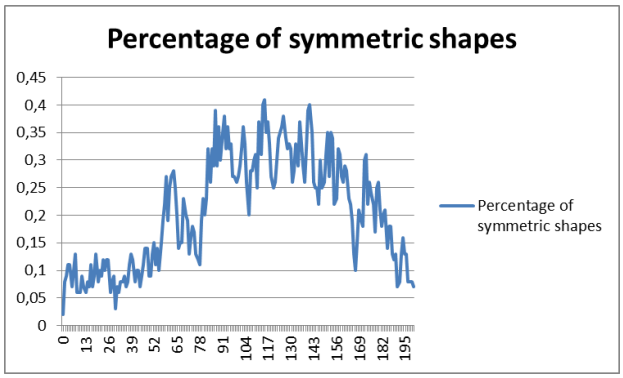


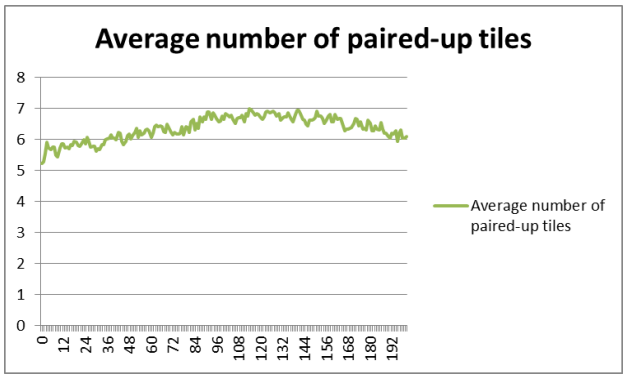
Experiment 10– Mutation rate: 0.1





Experiment 11– Mutation rate: 0.2

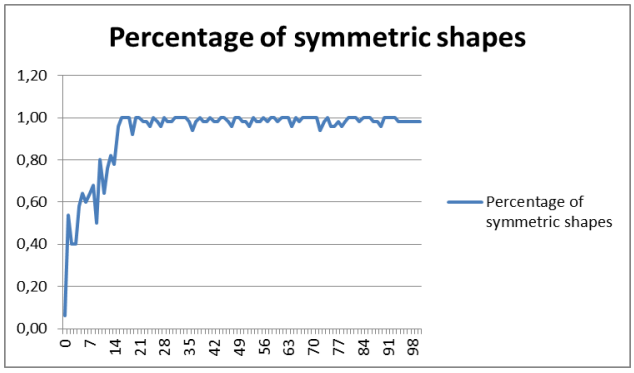


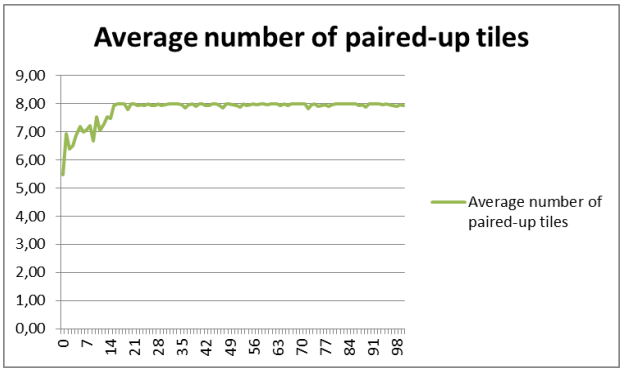


Set of experiments 12-13-14

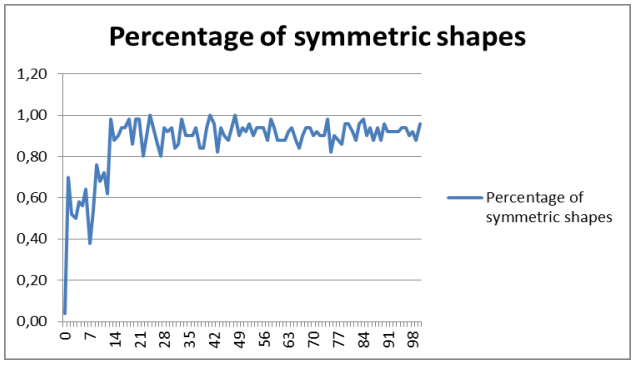
- Fitness: 1 if the produced shape is symmetric, 0 otherwise
- Number of generations: 100
- Population size: 50

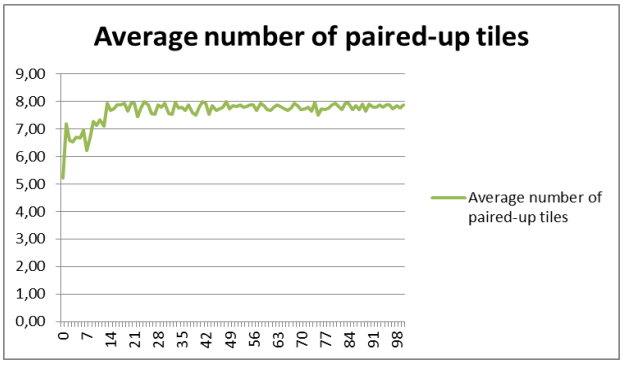
Experiment 12– Mutation rate: 0.02



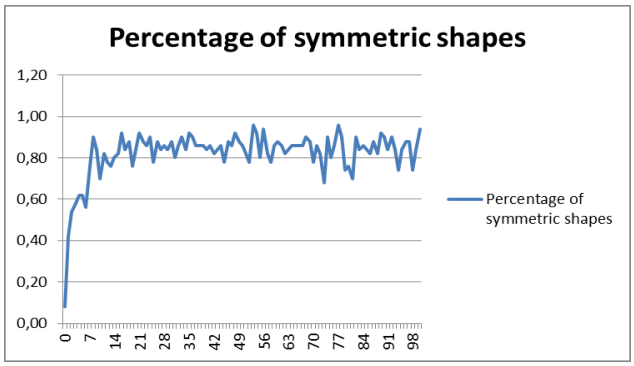


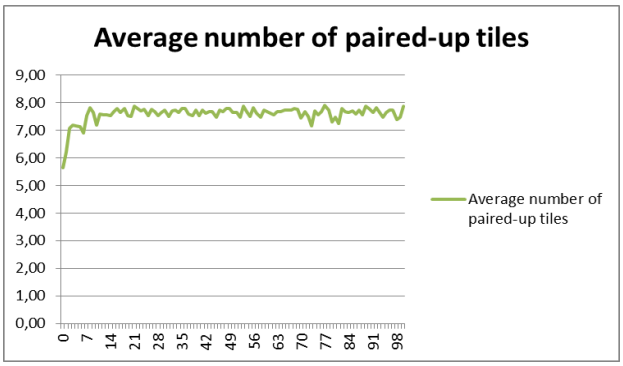
Experiment 13– Mutation rate: 0.1





Experiment 14– Mutation rate: 0.2

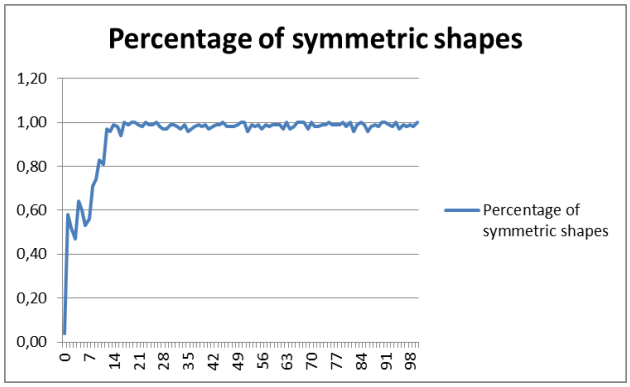


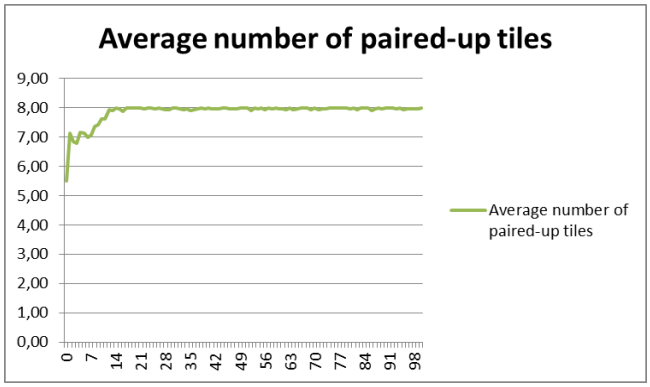


Set of experiments 15-16-17

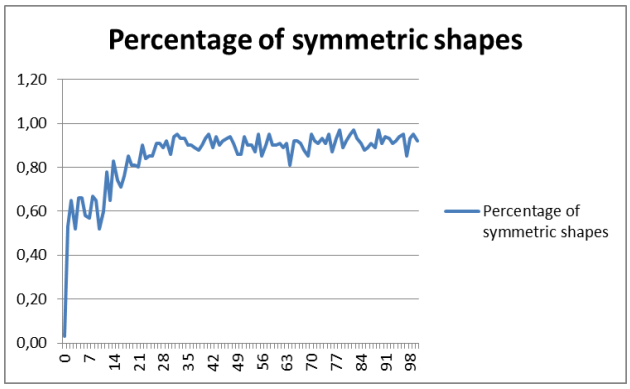
- Fitness: 1 if the produced shape is symmetric, 0 otherwise
- Number of generations: 100
- Population size: 100

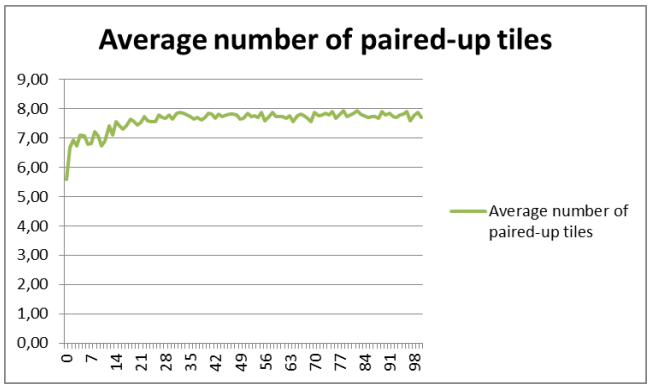
Experiment 15– Mutation rate: 0.02



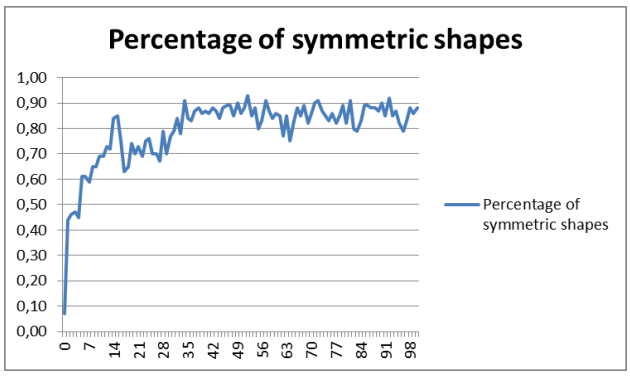


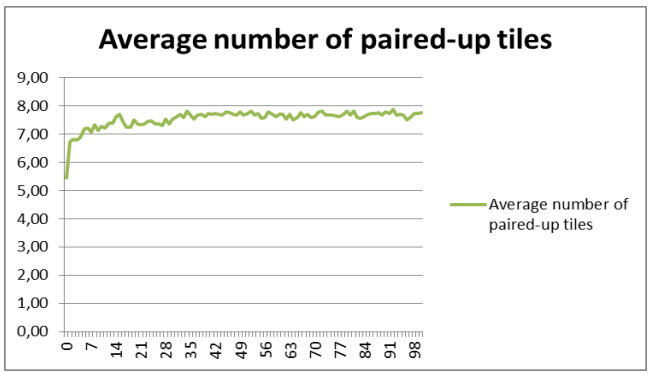
Experiment 16– Mutation rate: 0.1





Experiment 17– Mutation rate: 0.2

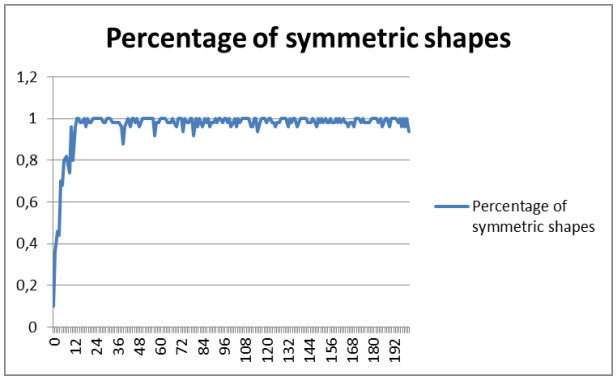


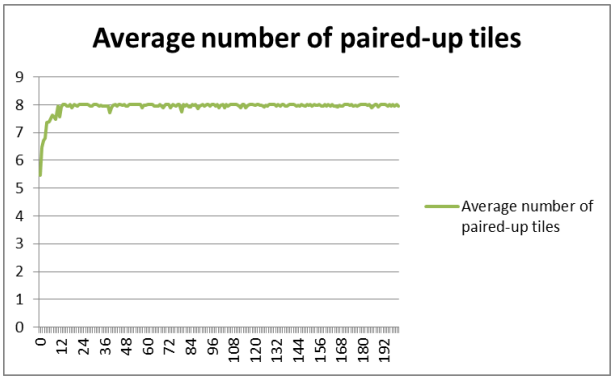


Set of experiments 18-19-20

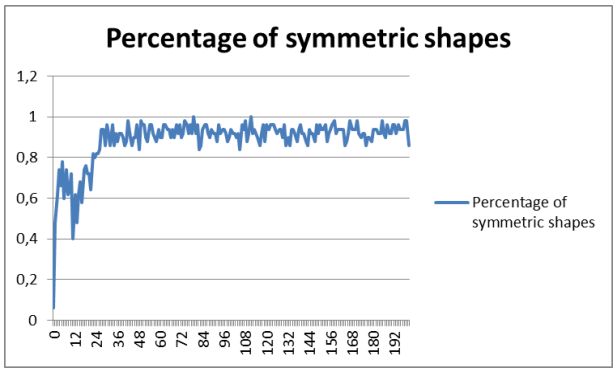
- Fitness: 1 if the produced shape is symmetric, 0 otherwise
- Number of generations: 200
- Population size: 50

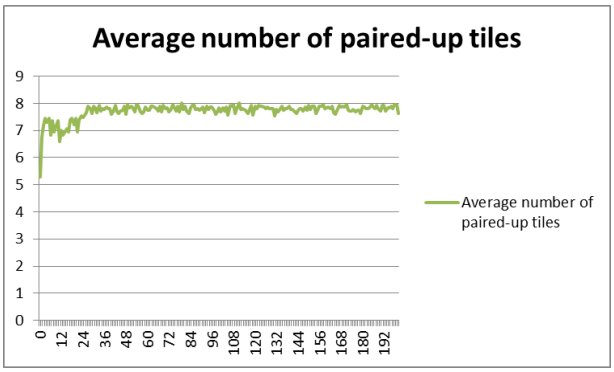
Experiment 18– Mutation rate: 0.02



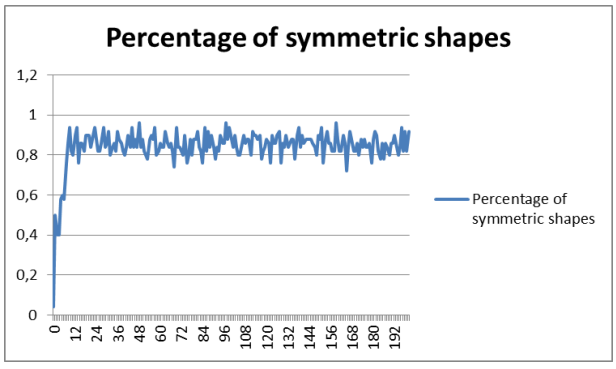


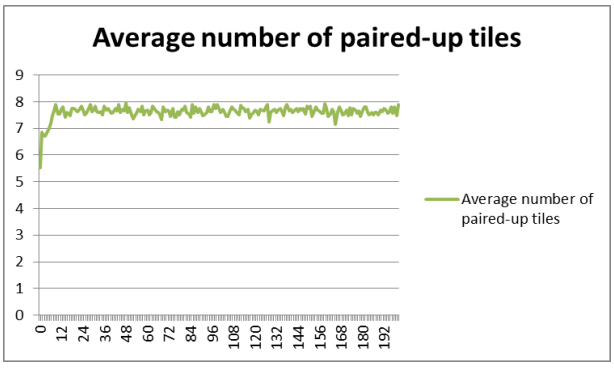
Experiment 19– Mutation rate: 0.1





Experiment 20– Mutation rate: 0.2

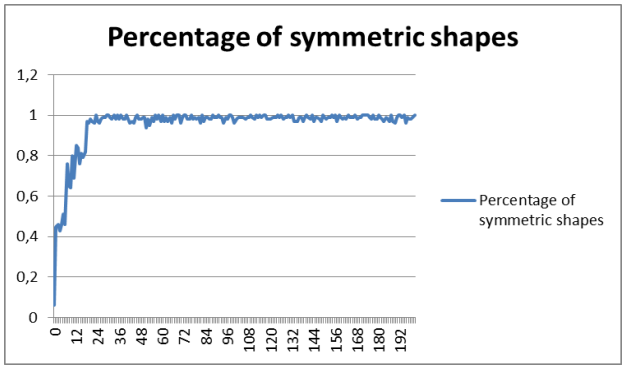


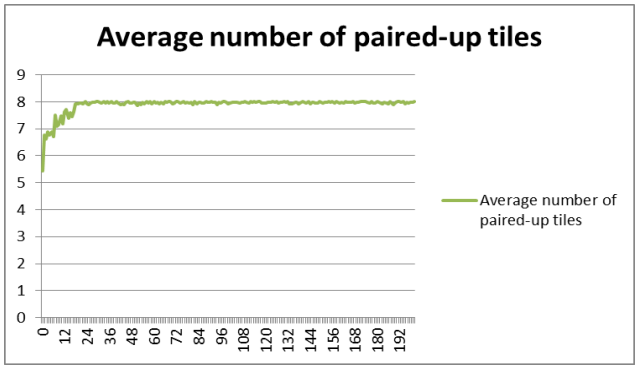


Set of experiments 21-22-23

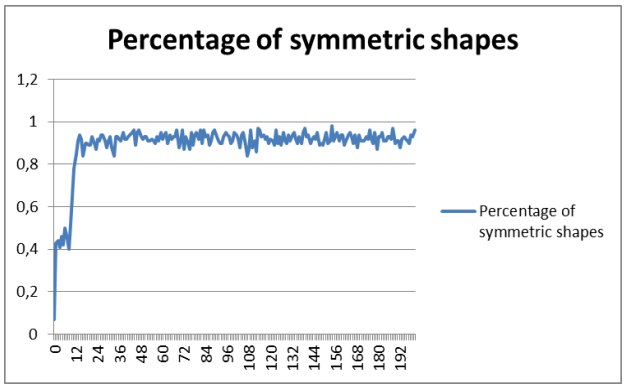
- Fitness: 1 if the produced shape is symmetric, 0 otherwise
- Number of generations: 200
- Population size: 100

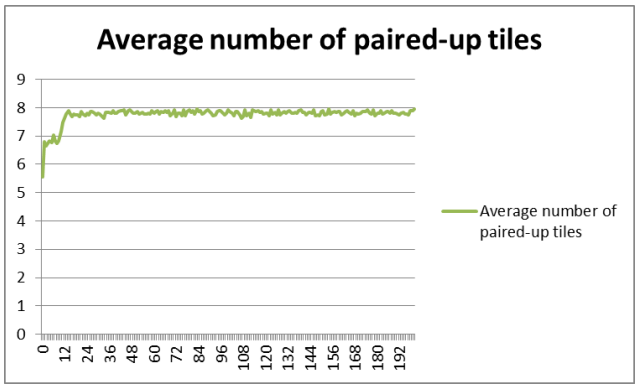
Experiment 21– Mutation rate: 0.02



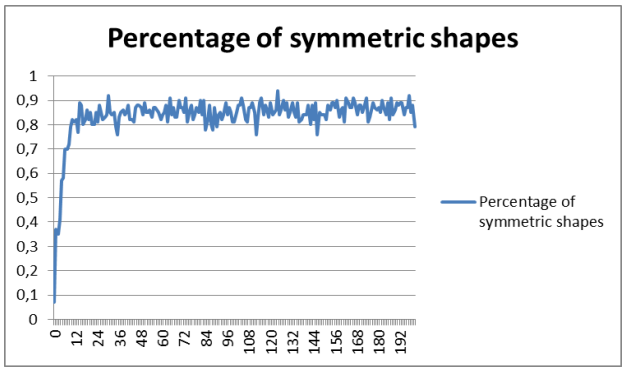


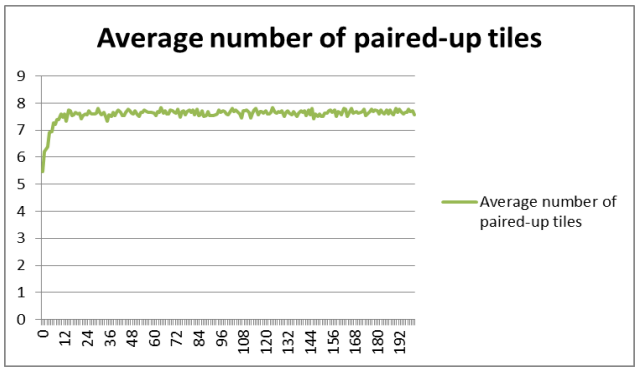
Experiment 22– Mutation rate: 0.1





Experiment 23– Mutation rate: 0.2

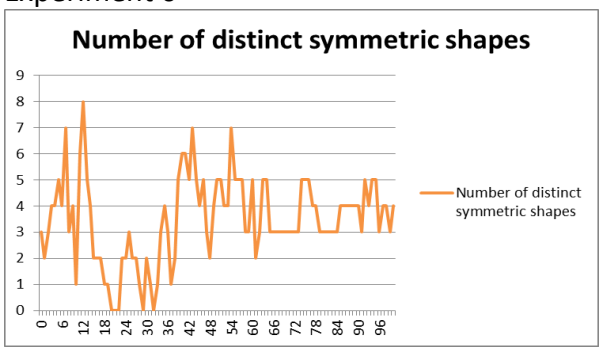




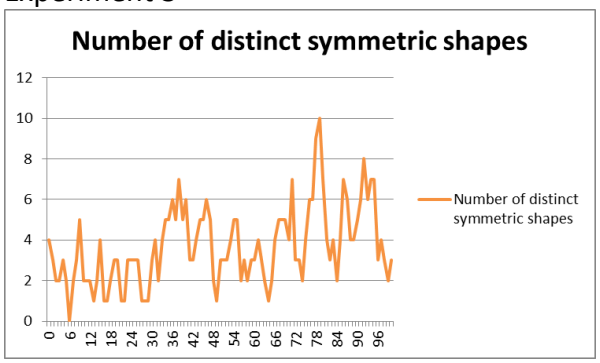
In light of these results, we can discard the experiments 1, 2, 4, 5, 7, 8 and 11 since they do not seem to produce more symmetric shapes as generations evolve.

Now we illustrate the number of distinct symmetric shapes produced in each generation of every successful experiment. Equivalent shapes that differ only in their position are taken also as different.

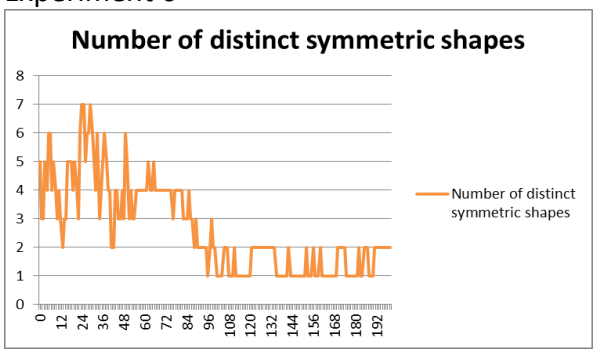
Experiment 0



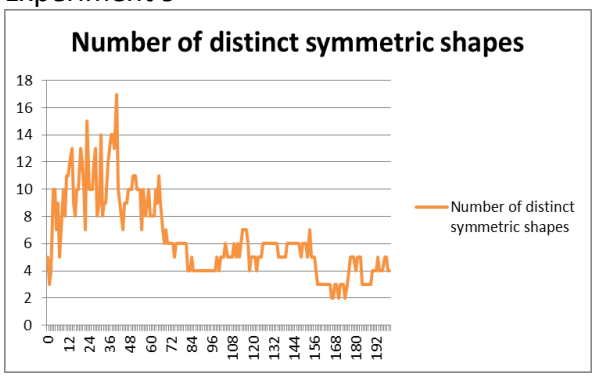
Experiment 3



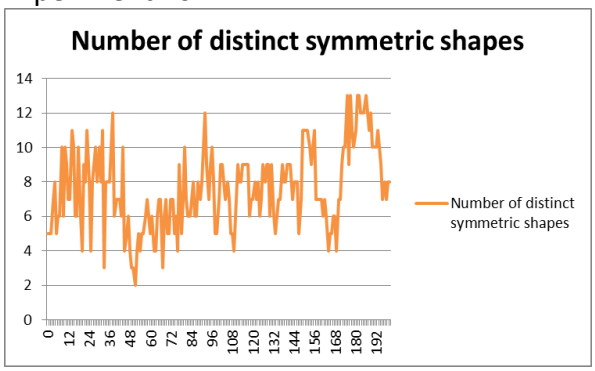
Experiment 6



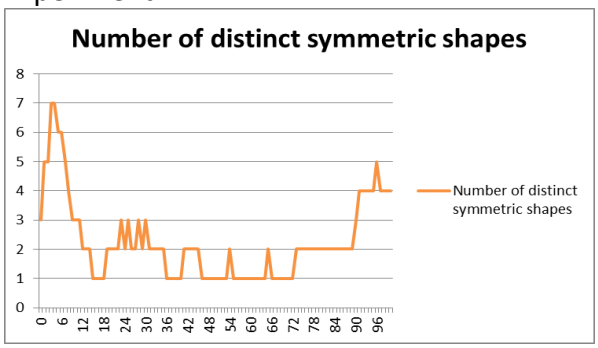
Experiment 9



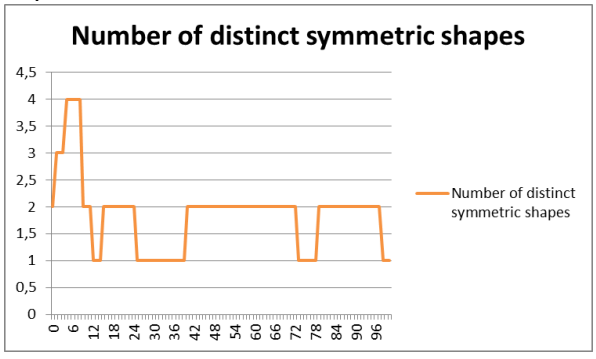
Experiment 10



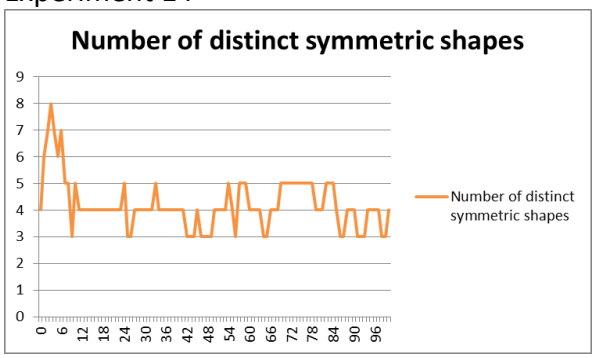
Experiment 12



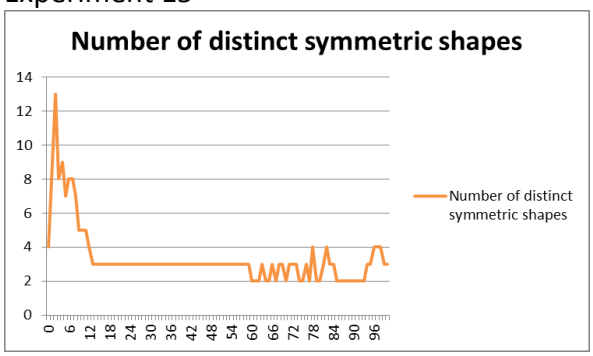
Experiment 13



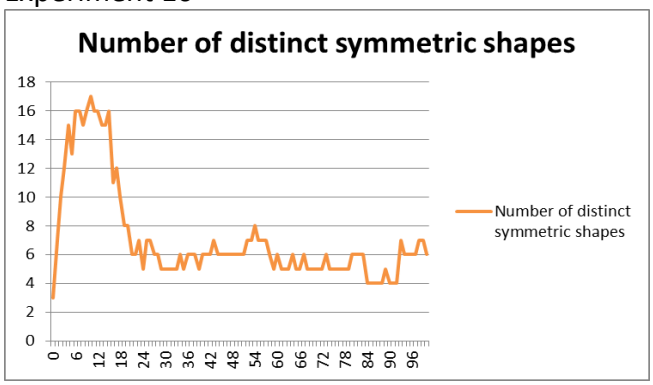
Experiment 14



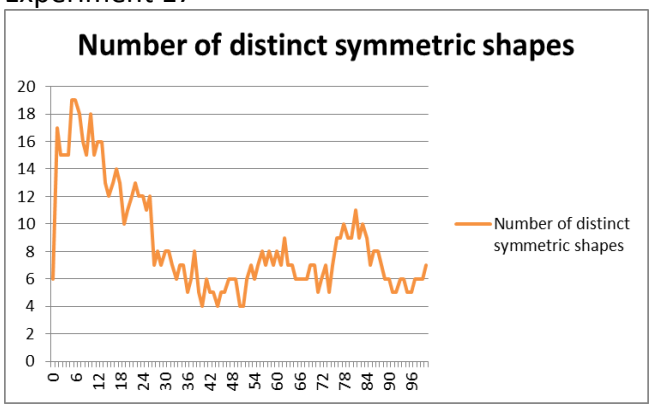
Experiment 15



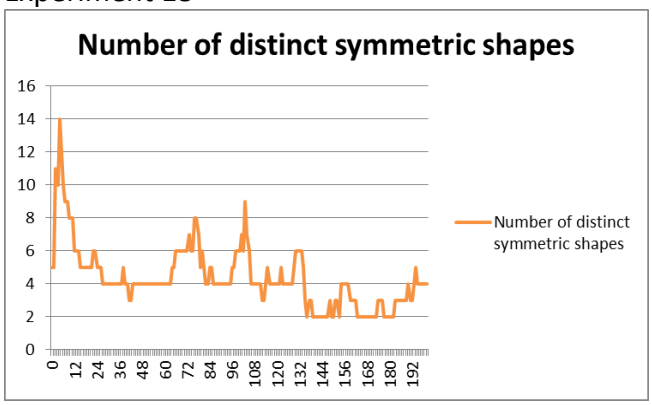
Experiment 16



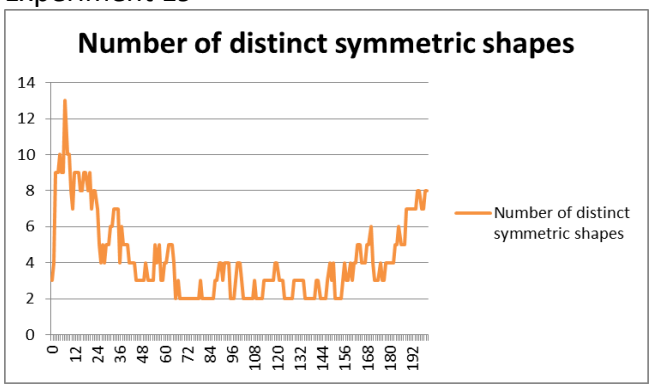
Experiment 17



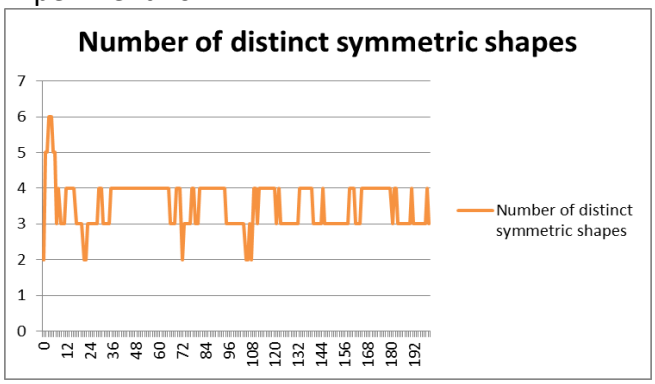
Experiment 18



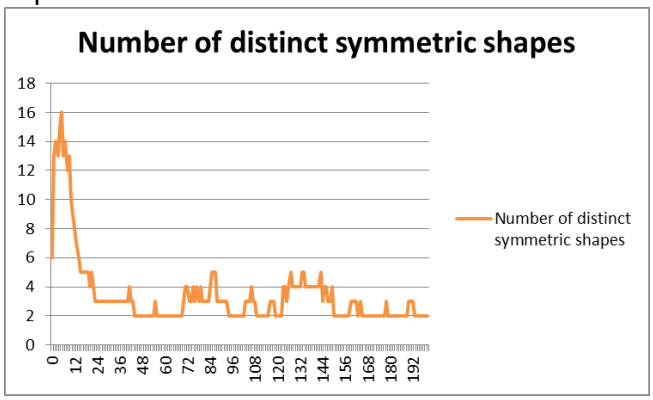
Experiment 19



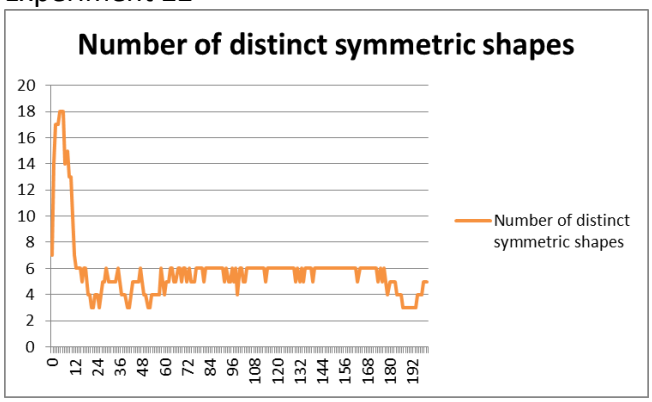
Experiment 20



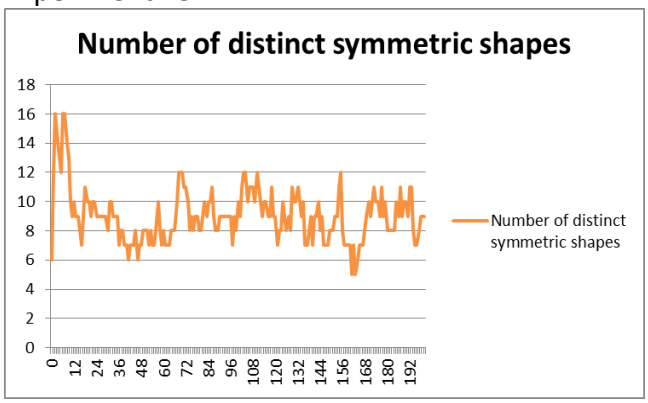
Experiment 21



Experiment 22



Experiment 23



Finally, we gather the execution times of each experiment in table 2.

#Experiment	Execution time
0	254.293545 s
1	247.318145 s
2	276.788831 s
3	550.452484 s
4	530.521344 s
5	518.161637 s
6	507.003999 s
7	503.281786 s
8	525.059032 s
9	996.665006 s
10	938.850699 s
11	969.454604 s
12	242.265856 s
13	253.028473 s
14	246.371091 s
15	487.396878 s
16	435.945935 s
17	466.552685 s
18	447.044569 s
19	461.119375 s
20	474.021112 s
21	931.415274 s
22	895.613226 s
23	917.210462 s

Table 2: Experiment execution times

5. Conclusions

Our objective is to produce vertically symmetric shapes of 8 tiles. With the help of a genetic algorithm, we evolve populations of rule sequences (that corresponds, actually, to shapes) in order to maximize this objective function.

In the results of the experiments we can see that the use of low mutation rates turns into a faster improvement of this objective and, eventually, into a stabilization of the percentage of symmetric shapes that coexist inside the population on a value next to 100%.

When using higher mutation rates, this percentage is not that high. Nevertheless, the genetic diversity is favoured, and thus there are usually more distinct symmetric shapes inside each generation.

Moreover, when we use the binary fitness approach, the process of obtaining a majority of symmetric shapes in each population is much faster (it is managed in less than 20 generations).

Also, in light of the number of distinct symmetric shapes inside each generation, it seems that in many cases, as generations succeed, the action of genetic operators homogenizes the populations.

References

[1] Stiny, G. (1980): Introduction to shape and shape grammars. Environment and Planning B, Vol. 7, No. 3. (1980), pp. 343-351

[2] Goldberg, D. (1989): Genetics Algorithms in search, optimization, and machine learning. Addison-Wesley.

[3] Gero, J.S., Suchil, J.L and Kundu, S. (1994): Evolutionary Learning of Novel Grammars for Design Improvement. Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, Vol. 8 (1994), pp. 83-94