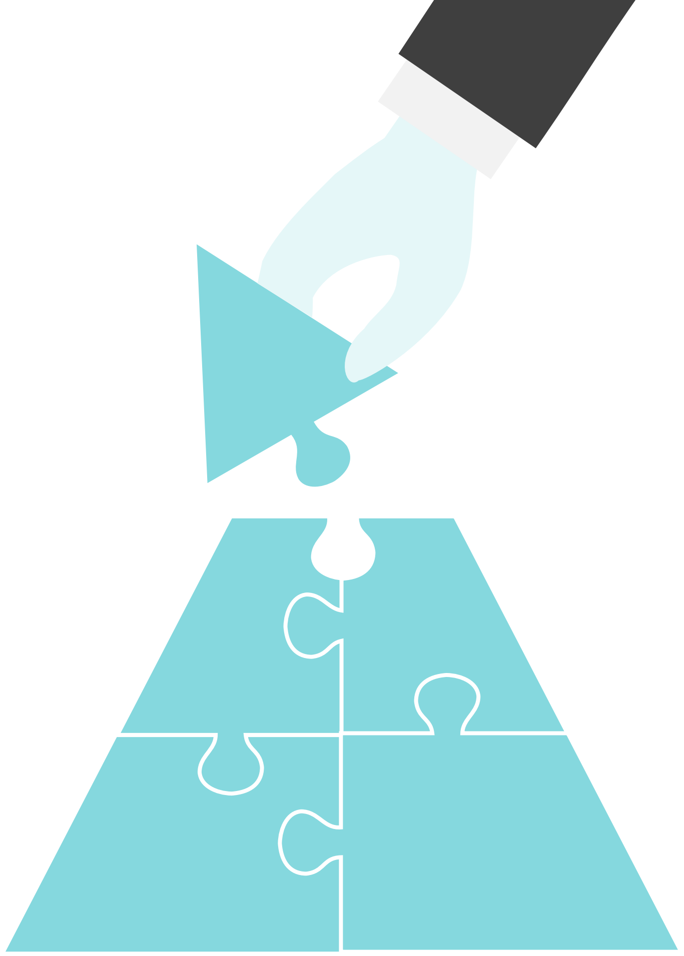
Mini Project

TOPICS IN BIO-INSPIRED COMPUTING

Introduction

Japanese puzzles - Nonograms is a logic puzzle with simple rules and challenging solutions.

The rules:
Your aim in these puzzles is to colour the whole grid in to black and white squares or mark with X. Beside each row of the grid are listed the lengths of the blocks of black squares on that row. Above each column are listed the lengths of the blocks of black squares in that column.
These numbers tell you the blocks of black squares in that row/column. So, if you see '10 1', that tells you that there will be a block of exactly 10 black squares, followed by one or more white square, followed by a single black square. There may be more white squares before/after this sequence. In our project '#' marks black and '`' marks white.



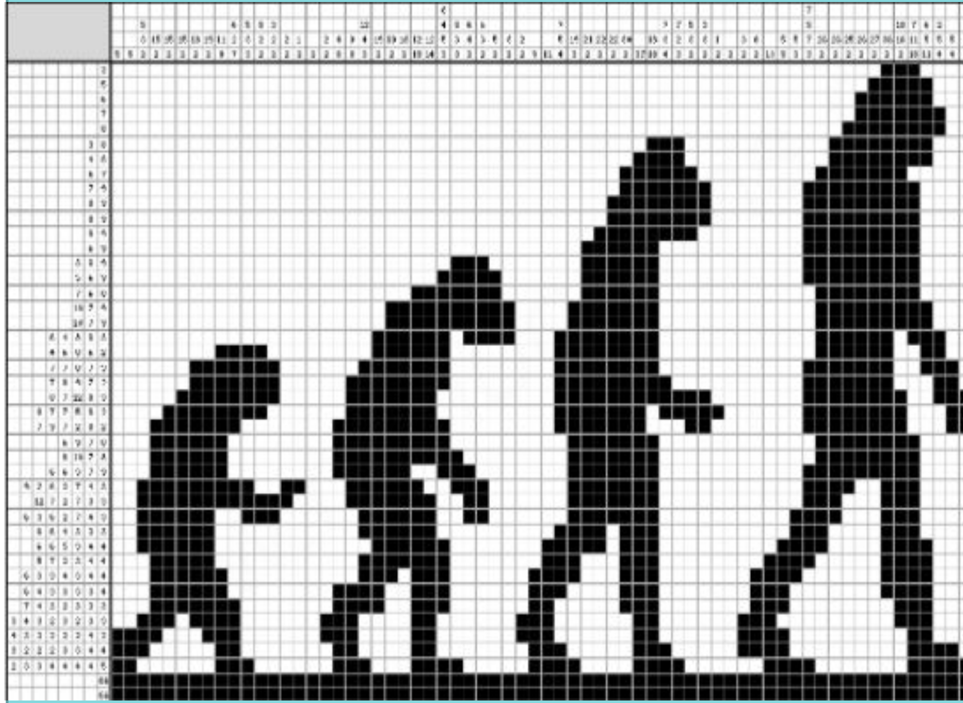
Introduction

While researching the topic of evolutionary algorithms we found that it is commonly used in the domain of problems with huge span of solutions.

Therefore we decided to use evolutionary algorithm in order to solve the famous nonograms problems.

Nonograms puzzles are **NP-complete** problems, hence they have a large number of solutions and its computational complexity is large as well.

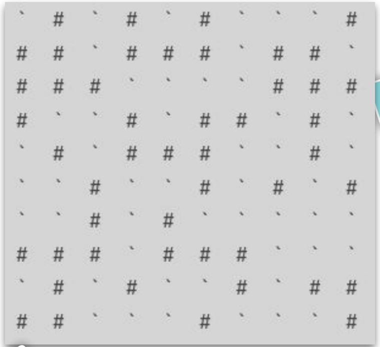
Our goal is to find an evolutionary algorithm that solves the nonogram puzzle problem with high precision approximation.



Method

Program Setup and Data Structures:

- **Nonogram**
  - Two 2D int arrays to represent the clues
  - Two integers, representing the dimensions
- **Solution**
  - Integer representing the fitness
  - Chromosome - bit string representing the solution itself as a flat matrixthe solution
- **Solver**
  - All additional information such as the mutation and crossover probabilities, number of generations, etc…
- **Visual Representation**
  - In a solution, the puzzle matrix is in a bit string representation (1's and 0's), but in order to see the solution, we represented it in #'s and `'s



```
` # ` # ` # ` ` ` #
# # ` # # # ` # # `
# # # ` ` ` ` # # #
# ` ` # ` # # ` # `
` # ` # # # ` ` # `
` ` # ` ` # ` # ` #
` ` # ` # ` ` ` ` `
# # # ` # # # ` ` `
` # ` # ` ` # ` # #
# # ` ` ` # ` ` ` #
```

A

Method

For each run the flow of the algorithm is as follows:

B

1. generate a population where each individual is a random filled puzzle, representing a solution, obviously not necessarily a correct one
2. for each generation:
   a. selection of 2 parents with highest fitness out of 10 randomly selected individuals.
   b. 50%/100% chance of crossover between these 2 parents
   c. 5% chance of mutation to the new individuals
3. iterating the population and keeping the one with the highest fitness
4. print the solution with the highest fitness

C

D

Experimental Setup

Crossover

We tested 2 different crossover methods:

1. Uniform Crossover - creating an offspring that consists of randomly picked bits from one of 2 parents.
2. Single-Point Crossover - randomly picking a point in the parents gene and switching between the apodosis of the parents (final version).

Mutation

For mutation we chose a random bit in the gene and switched it from ‘0’ to ‘1’ or from ‘1’ to ‘0’ accordingly.

Selection

We chose tournament selection method:
Picking the two parents with best fitnesses out of 10 random individuals from the population.

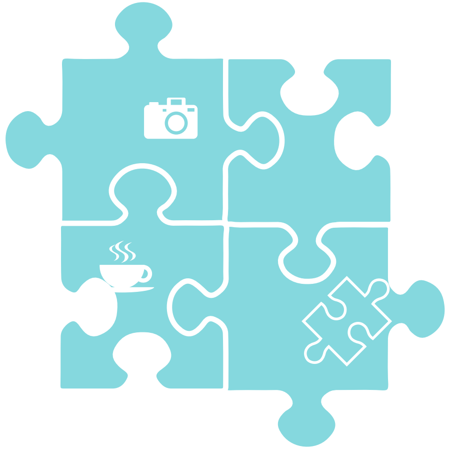
Experimental Setup

Fitness Function:
We made 2 different fitness functions that present two different approaches to progressing towards solving the puzzle:

F1 - Fitness function that is based on the amount of rows and columns whom satisfy the clues - for each row and column that satisfies the clue the fitness increases by 1, up to a max of (rows+columns).

F2 - Fitness function is based on the satisfactory problem, replacing the AND-operator with the AVG-operator, and replacing the OR-operator with the MAX-operator, resulting in a max fitness of (rows+columns)

Both fitness functions suggest that the bigger the fitness an individual has, the better it is for the evolution.



Experimental Setup

We ran our experiment on puzzles of sizes 5x5 and 10x10.
The program can receive any size of puzzle, these were our picks for the experiment.

We ran our experiment with different sizes of population and generations.

We also ran it using different chances for mutation and crossover.

Lastly, we tested our algorithm using the two different fitnesses mentioned earlier.

In the following slide we'll present the results and compare them with randomly generated solutions.

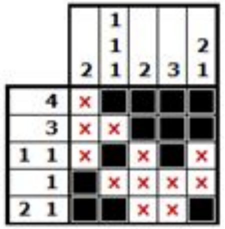
Each test is the result of 10 runs with the defined settings.

We marked board 1 with fitness function 1 'A', board 1 with fitness function 2 'C'.
We marked board 2 with fitness function 1 'B', board 2 with fitness function 2 'D'.

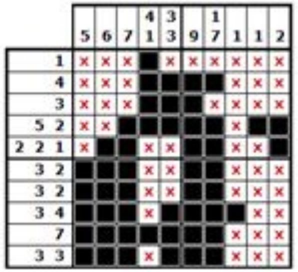
For each board and fitness function we ran the algorithm 3 times with different evolutionary operators and different population size -

1. random generated solution
2. different probabilities of crossover

Board 1



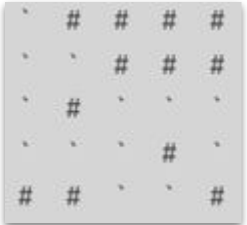
Board 2



Results - Fitness 1

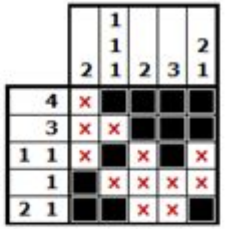
Test	Puzzle	Population	Crossover	Fitness Avg.
A-1 (Random solution)	5x5	-	-	4/10
A-2	5x5	100	50%	4.6/10
A-3	5x5	100	100%	4.2/10
A-4	5x5	1000	50%	6.1/10
A-5	5x5	1000	100%	6.2/10
B-1 (Random solution)	10x10	-	-	2/20
B-2	10x10	100	50%	1.9/20
B-3	10x10	100	100%	1.9/20
B-4	10x10	1000	50%	3.5/20
B-5	10x10	1000	100%	3.3/20

A-4 individual



```
.  #  #  #  #
.  .  #  #  #
.  #  .  .  .
.  .  .  #  .
#  #  .  .  #
```

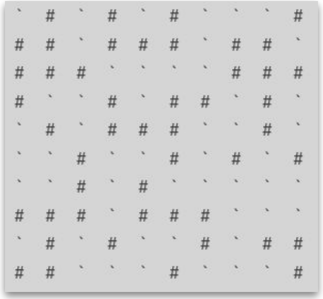
Solution



Results - Fitness 2

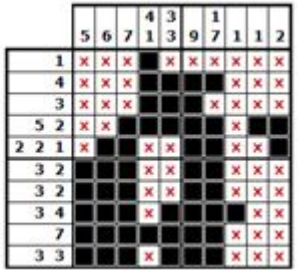
Test	Puzzle	Population	Crossover	Fitness Avg.
C-1 (Random solution)	5x5	-	-	8.93/10
C-2	5x5	100	50%	9.25/10
C-3	5x5	100	100%	9.24/10
C-4	5x5	1000	50%	9.48/10
C-5	5x5	1000	100%	9.53/10
D-1 (Random solution)	10x10	-	-	17.91/20
D-2	10x10	100	50%	18.64/20
D-3	10x10	100	100%	18.49/20
D-4	10x10	1000	50%	18.73/20
D-5	10x10	1000	100%	18.87/20

Random

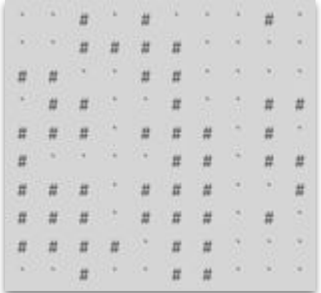


```
` # ` # ` # ` ` ` #
# # ` # # # ` # # `
# # # ` ` ` ` # # #
# ` ` # ` # # ` # `
` # ` # # # ` ` # `
` ` # ` ` # ` # ` #
` ` # ` # ` ` ` ` `
# # # ` # # # ` ` `
` # ` # ` ` # ` # #
# # ` ` ` # ` ` ` #
```

Solution



D-5 individual



```
` ` # ` # ` ` ` # `
` ` # # # # ` ` ` `
# # ` ` # # ` ` ` `
` # # ` ` # ` ` # #
# # # ` # # # ` # `
# ` ` ` ` # # ` # #
# # # ` # # # ` ` #
# # # ` # # # ` # `
# # # # ` # # ` ` `
` ` # ` ` # # ` ` `
```

Conclusion

Our results suggest that the success of the evolutionary algorithm is dependent on a number of things:

1) Size of the initial popularity
2) Finding a suitable fitness function to the problem
3) The larger the span of solutions - the better for the evolutionary algorithm

It seems that our first fitness approach hasn’t succeeded as we were hoping, but after adjusting our fitness function, we got a slight improvement over a random generated solution.
These minor improvements are sticking out greatly in the larger puzzle which strengths the suggestion that evolutionary algorithms should commonly be used in problems with huge sets of solutions.

Interestingly, we noticed that when we ran the algorithm with 100% chance for crossover the results were better.

Also, we saw that increasing the number of generations didn’t produce significant better results, and we excluded it from the report, but it can be accessed in the attached excel file that details of all of the runs we performed for the project.

Thoughts on Improvement

For future improvements we would like to make a better fitness function that will result in an adequate solution within a certain amount of generations, as now it seems we reached a limit in our fitness that the evolution can't exceed.

We would also like to set a max fitness breakpoint that will imply that once reached the solution there's no need to go on with the evolution.

We would like to add a different crossover function that is more rows/columns focused.

Thank You

Liad Gabay
315855940

Dor Gal
308575588

Yuval Dahan
316583749

Appendix

Sources of information:

1. Kenneth A. De Jong and William M. Spears. Using genetic algorithm to solve np-complete problems. In Proc. of the Third Int. Conf. on Genetic Algorithms, pages 124–132, 1989
2. Wouter Wiggers. A comparison of a genetic algorithm and a depth first search algorithm applied to Japanese nonograms

Fully Detailed Report:

1. Download
2. Google Sheets

Code:

1. Download
2. Github

Appendix

School Schedule problem

The previous problem that we tried to solve using an evolutionary algorithm is the School Schedule problem - finding the best school schedule for an elementary school. The school would provide constraints for the teachers, and weekly hours for the fields of study, and the algorithm would create a full school schedule.

Hours & Day Encoding

	Sun	…	Thu	Fri
8	0	...	32	40
9	1	…	33	41
…	…	…	...	...
15	7	...	39	47

Bit-String Representation

Id	Field	Time	Class	…	Time	Class

- Every 16 bit represents a teaching slot.
  - The first 4 bit represent the teacher id
  - The next 2 bit represent the field she is teaching
  - The next 5 bit represent the encoded time slot
  - The next 5 bit will represent the class number

A representation of a solution is a long bit-string composed of alot of teaching slots.

Crossover

exchanging random number of slots from two parents

Mutation

selecting a random teaching slot and randomise it

Selection

selecting 10 random solutions and selecting the one with the highest fitness

Our fitness function needs to be Multi-Objective as we have more than 1 goals to consider:

- We need to minimize the number of violating a teacher constraints.
- We need to minimize the number of overlapping between teachers so every teacher overlapping another teacher (same class in the same time).
- We need to minimize the number of internal overlapping (same teacher have two different classes in the same time.
- Every class should have a certain hours to each fields, so we needed to maximize fields hours.
- Classes should not have more than 2 consecutive hours in the same field.