Computational Biology – EX2

Eidelman Elli – 306679499

HowTo:

My solution was written in python and “compiled” to exe with py2exe lib, due to the usage of many GUI libraries, exe is quite big, I’ve uploaded it to my github that can be found here:

<https://github.com/shushkis/pythonProjects/raw/CoronaSim/comp_bio_ex2/Futoshiki_GUI.7z>

To run just double click on Futoshiki\_GUI.exe you will get this prompt that tells you that you will need to choose a puzzle file and that the keys are F1 – regular genetic algorithm, F2 – Darwin, F3 – Lamarck, F4 – recursive, Tab – for random board, r – reset

At the end you will get a graph with indication that solution found in X seconds or didn’t find any solution in X seconds and a graph of fitness over generations.

Code review:

I’ve implemented a genetic algorithm, where solution is a NxN matrix with 1-N numbers. First generation is chosen randomly next generations will be as follows best solution survives for next generation other solution are again chosen randomly where we prefer solution with good scores. I’ve tried to run with different amount of population, as suggested in task,100, 50 and 500. Main difference I saw is the faster conversance to solution or to a local maximum when I’ve less population number.

Main loop – solve:

Runs until solution found, fitness is equal to max constrains or to maximum of iteration (set in constants).

Fitness - Is calculated as follows, for each line check that line has one instance for each 1 – N if not punish by how much missing numbers, maximum score – N

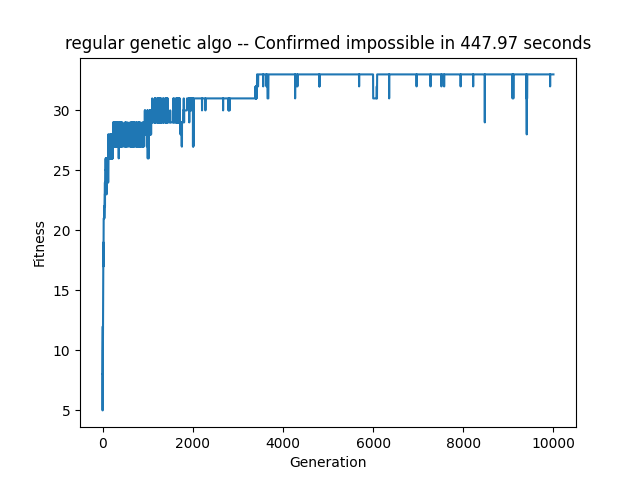
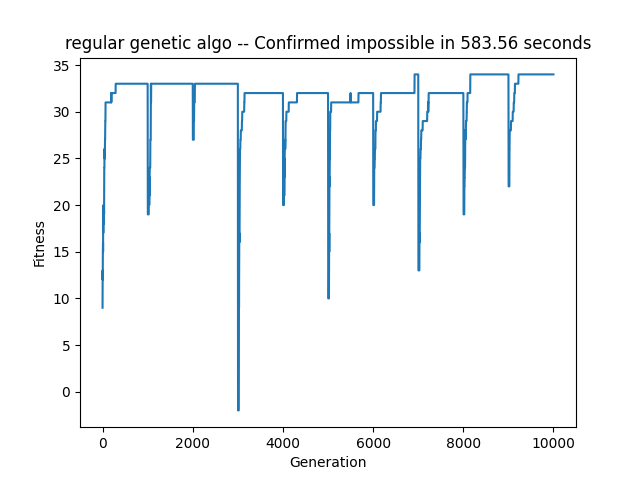
Same for columns, maximum score – N

For each cell check that it is ok with inequality sign, punish if not.

For each given numbers by user, check if location and value is matched in solution if not punish max -9999.

Maximum score is N+N+ N\*N = 2N+N2 and this is the stop condition in main loop.

For each random solution in next generation, we can apply both crossover function and mutation function depend on the probability that it’s alternating between low and high depends if we think that we are on a local maximum, which I’ve implemented by asking how many times the same best fitness appeared. I also make a force change in probabilities every 1000 generations for 10 generations to force refresh the population.

Unfortunately, in most cases ends up with dead-end. Like in next figure

For that I’ve implemented several things to try and fight this phenomenon.

First, when probability to morph is set to high, I make the crossover function for each new solution to mate with my best solution to try and spread its genes around the population, some kind of force selection. I also set a more in favor crossover point for the best solution parent.

For mutation part, when probability set to low. I change the only cell with very low probability.

But when it is set to high, I reshuffle the whole line in solution.

I also have a hail marry solution when I’ve a large percentage of aberrations in best solution population and I didn’t find any solution I will kill the whole generation and start new one.

I’ve found out that if I give a strong advantage to the best solution and through away the worst, I end up with premature convergence **I’ve added the worst** solution when my loop is stuck in maxima.

Graphical user interface, application

Description automatically generated  
  
added some selection heuristics.

Graphical user interface

Description automatically generated

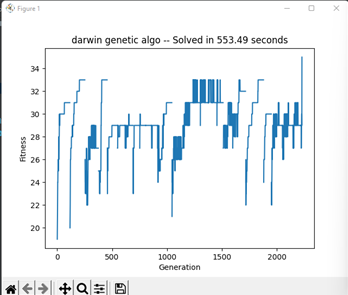
Added kill all

A picture containing text

Description automatically generated

7X7 – hard grid

Chart, line chart

Description automatically generated

I’ve run the three algorithms on few boards, some random some easy some hard in different sizes 5x5, 6x6 and 7x7.

I’ve ran all 3 several time, each iteration takes some time, probably needed to multiprocess here. Here is a table of some features that I compared.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Found any solution | Found solution for each board | Convergence issues | Larger board makes no solutions | More inequalities makes no solutions | Best Found solution in |  |
| Regular | Yes | No | Local maxima | No | Yes | 850 |  |
| Darwin | Yes | No | Local maxima | No | No | 2250 |  |
| Lamarck | Yes | No | Not converging | No | Yes | 500 |  |

Possible mitigations to overcome convergence issues:

Regular – I think that I did my best to overcome this issue, maybe what I can do is to change a bit my fitness function to punish more when the inequalities are bad like I punish for the wrong place of given numbers.

Darwin – it seems that I need to add more bad solutions here, I did add the worst one when I’m in high probability to morph, maybe I need to add some more because the optimization is very harsh, it places the given numbers to the right place which is my hardest punishment in the fitness function.

Lamarck – I think that I need to select just part of the population to optimize I’m very fast converging to solution if I find it, but if not in most cases the algorithm just won’t coverage.

Concluding the above:

All three are very randomly to predict if will end-up with a solution or not.

I need to run in parallel few boards over the night to get any statistics and when running on the same board repeatedly I get very different results, from converging in less then 1000 generations to running 1M and not converging at all.

Darwin’s solution seems to be more likely to find any solution. it converged in most runs.

The fastest one is Lamarck although in total most cases it just won’t find any. But when it did it was quite fast.

The regular algorithm was very sporadic of the three, I don’t have any good bottom-line about the mean it is converged nor the mean of how many generations it takes.