1. Introduction:

Genlib project – Our built library for GA. Contains Chromosome class implementation, Genetic model, and Functions Base classes.

8Queen project – Contains GA and brute-force for 8 queen problem.

TSP project – Contains GA and greedy algorithm for TSP problem.

In our code, we created our own Genetic Model. This model receives:

Initialize Function

Selection Function

Crossover Function

Mutation Function

The model runs the general GA flow with given functions. The model doesn't need to know what each function does, since each Inherits its own base class (Initialize Base, Crossover base …), which have a virtual method run () which the model invokes.

We also created a chromosome class that manages itself.

Eight queen problem

Chromosome representation: size 8 chromosome, index is row number and cell value is column number

Initialize function: We create a set of 8 number 0-7 and each time choose randomly a number, put it in chromosome, and take it out of set. With this initialize function, 2 Queens cannot be on same row on column, thus we're starting off with higher fitness function.

Selection: Exploit explore selection. With some probability we choose either to explore (Strong with random chromosome) or exploit (strong with next strongest chromosome). We also define a selection

ratio (number between 0 to 1) to define how much couples do we want (0 == 0 couples, 1 == population size – 1 couples).

Crossover – uniform crossover. If with a given parent, we get a chromosome that we already have, we choose from the second parent. If we reach a deadlock (both gens exist in child chromosome), we will choose randomly from a set of genes not chosen. Thus, we receive a stronger chromosome (2 queens cannot be on same row on column).

Mutation – We swap 2 genes in chromosome with given probability.

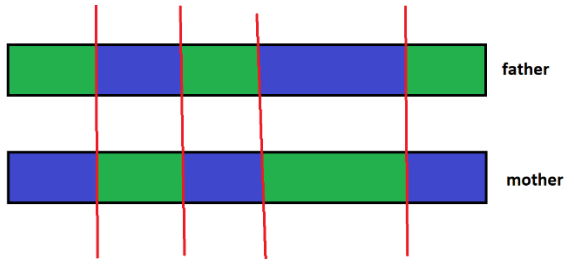
TSP problem

Chromosome representation: size 48 chromosome, each cell defines a city we traveled to

Initialize function: We create a set of 48 number 0-47 and each time choose randomly a number, put it in chromosome, and take it out of set. With this initialize function, we visit once, so each chromosome is a legal chromosome.

Selection: Roulette Wheel Selection. We sum up all chromosome's fitness, then we select a random number between 0 to sum. We create a partial sum and sum all fitness until it's greater or equal to the random number. When it's greater or equal, we choose the last chromosome that we summed its fitness. We do this again for the second parent. We also define a selection ratio (number between 0 to 1) to define how much couples do we want (0 == 0 couples, 1 == population size – 1 couples).

Crossover – 4-point crossover. We define 4 random points, and crossover as such:



First child gets the blue gens, the second child gets the green gens. Note: green may not always start at top chromosome, and maybe chose to be bottom (it decided randomly).

We tried using uniform crossover, but the fitness raised very slowly - if at all , and with single- and 2-point crossover the fitness raised faster , but once it finished raising, the mean diff from generation to generation remained 0. With 4-point crossover, we reached better results – higher accuracy and the mean diff from generation to generation once close to optimal fitness was reached was a low number, but not 0.

Mutation – We swap 2 genes in chromosome with given probability.

File naming and format

The file naming is made of:

Prefix_num1-num2-num3.txt

Num1 is population size, num2 is crossover probability, and num3 is mutation probability.

The GA file content is as such:

The first number – represents population size

The second number – represents crossover probability

The third number – represents mutation probability

The fourth num – represents time in seconds to finish

From row 4 until the end there 3 numbers num1,num2,num3.

Num1 - the best chromosome (cost\loss) in the recorded generation

Num2 – the worst recorded (cost\loss) chromosome in generation

Num3 – the mean recorded (cost\loss) in generation

User can decide per how much generations he wants to record data, and he can also control after how much generations to terminate the model.