Home Problem 2

Anton Karlsson, 930225-1179, antkarls@student.chalmers.se

October 11, 2017

FFR105, Stochastic Optimization Algorithms
Chalmers Tekniska Hgskola

Problem 2.1

a)

In the trivial case of 2 cities there is only one distinct path between them.
For the case of three or more cities we can generalise. Consider the case of N cities. Then there are N ways to pick the first city. Next, without replacement, there are $N-1$ possible ways to pick the second city. Continuing in this manner we arrive at a total of $N!$ ways to select a permutation of N cities.
However, since for instance the clockwise path between a set of citites is considered equal to the anti-clockwise path, we have to divide the total number of paths by 2.
Moreover, there are total of N ways to pick the starting node of any given path, and since we do not really care which node is the starting node in a complete circuit of cities we also have to divide by N. The final result is that there are $N!/(2N) = (N-1)!/2$ distinct paths. In conclusion:

$$\text{Number of distinct paths} = \begin{cases} 1 & \text{if } N = 2 \\ \frac{(N-1)!}{2} & \text{if } N \geq 3 \end{cases} \tag{1}$$

b)

A genetic algorithm was implemented as specified in the problem description. Swap mutation was used and no crossover was carried out. The program runs for 10^4 generations as default, and to run it simply run the script "GA21b.m".
A population size of 100 individuals was used. The mutation probability was set to 5 divided by the number of cities, in this case resulted in a mutation rate of 0.1. The tournament selection parameter was set to 0.75 and a tournamentsize of 2 was used. At every generation one copy of the best path was carried over to the next generation.

c)

The provided AntSystem.m was implemented as described. With notations following the textbook, α was set to 1, β was set to 3 and ρ was set to 0.5.

d)

The nearest neighbour path was generated with the program NNPathLengthCalculator. The program was executed 10^3 times which resulted in a average path length of 149.8723 length units and a standard deviation of 7.7432 length units.

e)

The genetic algorithm ran for 10^5 generations and the best path length was found to be 152.3551, which was found after about 96000 generations. This was not quite as good as the nearest neighbour path found in d). This is however still an amazing performance considering that only $10^7/(49!/2) \approx 3 \cdot 10^{-56}$ percent of the search-space was evaluated. The ant colony optimization algorithm found a path length of about 121.2 in a matter of seconds. After about 400 iterations the algorithm stopped finding better paths and the minimum path length was found to be 121.1052 length units. The specific path can be found in the folder "e" in the matlab file BestResultFound.m.

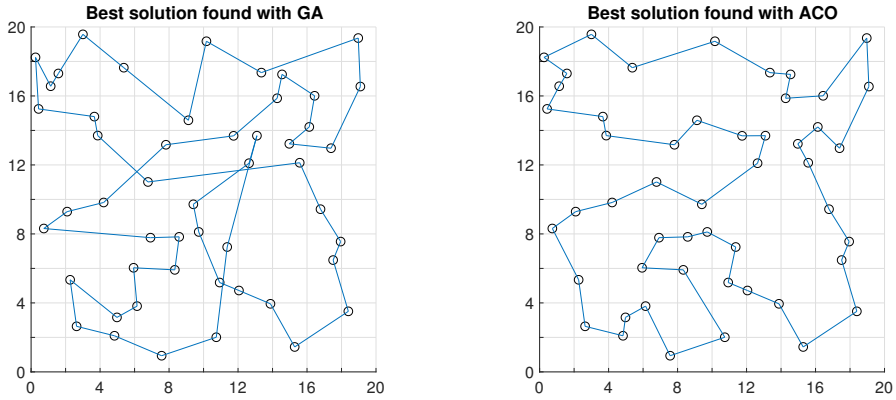

Figure 1: This figure visualises the best path in the traveling salesman problem with 50 cities found by GA and ACO. The GA path length is 152.4 length units and the ACO path length is 121.1 length units.

Both paths are visualised in figure 1. The conclusion is that ACO is very quick in finding the nearest neighbour thanks to the way the selection probabilities are defined, and the nearest neighbour path is in most cases significantly better that what you would get by simply randomly picking a path which is what you do initially in the GA.
As an experiment, the nearest neighbour path was generated and inserted in the population of a GA, and not so surprisingly the GA performance greatly improved, albiet still not to ACO levels.

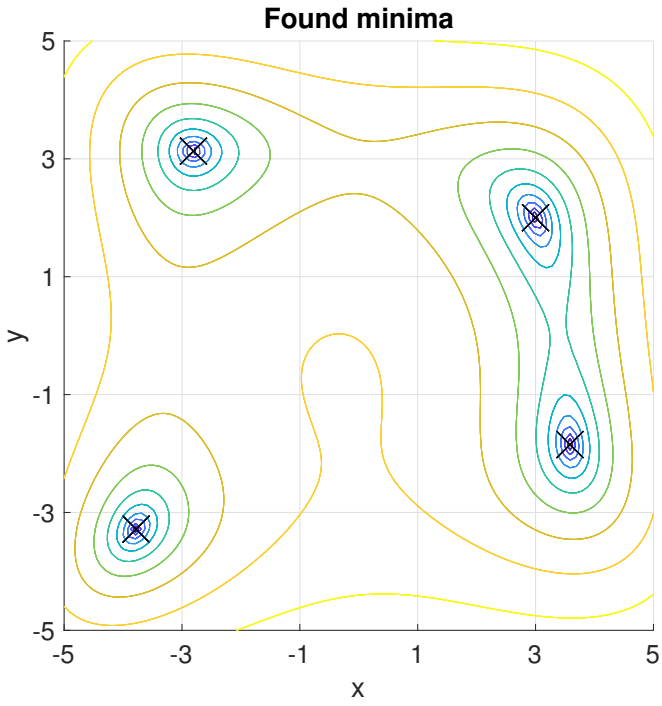

Figure 2: This figure shows the contour plot of the objective function with the corresponding minima, found by PSO, marked with a cross.

Problem 2.2

A standard PSO was used to find local minima of the given objective function in the range $(x, y) \in [-5, 5]$. A total of 40 particles were used with $c_1 = c_2 = 2$. Their speed was limited to 0.1 and their inertia-weight was initially set to 1.6, but was reduced each iteration by the multiplicative factor 0.99 until it reached 0.4 where it was kept fixed. 20 runs were performed in order to find all minima, each with 500 iterations, which resulted in four minima shown in figure 4 and specified in table 1. The algorithm performed very well, although the objective function is a bit simple, and in many cases fewer particles as well as fewer iterations would suffice. However, in order to increase the likelihood of the algorithm performing well when handed in the parameters were set a bit excessive.

Table 1: This table shows the numerical values of the minima found by PSO with six decimal precision.

x	y	$f(x, y)$
-3.779310	-3.283186	0.000000
-2.805118	3.131313	0.000000
3.000000	2.000000	0.000000
3.584428	-1.848127	0.000000

Problem 2.3

Individuals

For the sake of clarity (at least for the author) the individuals were implemented as Structs (hence the uppercase naming) with two fields, hiddenWeights and outputWeights corresponding to the counterpart in the feed forward neural network. This way the specific dimensions of the network didn't have to be passed to every function since that information is already stored in the weight dimensions. To make use of as much of the Matlab program written in the introductory workshop the function Chromosome2Indivudual and Individual2Chromosome were made to move between the two formats. Chromosomes in this case are just the concatenated and flattened weight matrices.

Evaluation

To evaluate each individual the braking system was used on all the slopes in the training set. For each run a "distanceScore", which corresponds to total distance divided by the slope length, and a "speedScore", which corresponds to the average speed divided by the max speed", was computed.
The fitness value corresponding to a specific slope i, denoted f_i, is the distance score times the speed score times 100. In this way the maximum fitness for a curve is 100 and the minimum fitness is 0. To compute the total fitness of an individual three additional parameters were introduced. The parameter "breakPenaltyFactor" specifies how much to penalise individuals that didn't make it to the end of the slope. In this case it was set to 0 meaning that these individuals have zero fitness. The fitness was then taken as the mean of the fitnessValues for all slopes in the training set times "meanFitnessFactor", c_μ, and was added to "worstFitnessFactor", c_w, times the worst fitness in the training set. This way a fitness of 100 translates to running the full length of the slope at the maximum .

$$f = c_\mu \cdot \text{mean}(f_i) + c_w \cdot \min(f_i) \tag{2}$$

In this case both c_μ and c_w were set to 0.5.

Other GA Operators

Tournament selection was used just like it has been for the previous exercises, with a tournament size of 5 and the tournament selection parameter set to 0.75. Likewise standard crossover was used with crossover probability of 0.8. Creep mutation was used a creep-rate of 1. The mutation rate was varied so as to keep the mutation rate high in the early stages to favour exploration, and followed the exact formula

$$P_{\text{mut}} = \frac{\max\{1, 20 \cdot 0.98^g\}}{m} \tag{3}$$

where m is the chromosome length and g is the current generation. The specific numerical constats were empirically found to perform well.
Note that the default number of generations was set to 50 in case the examinor wants to run the algorithm within reasonable time, if one were to actually train the network this would be set to a larger number.

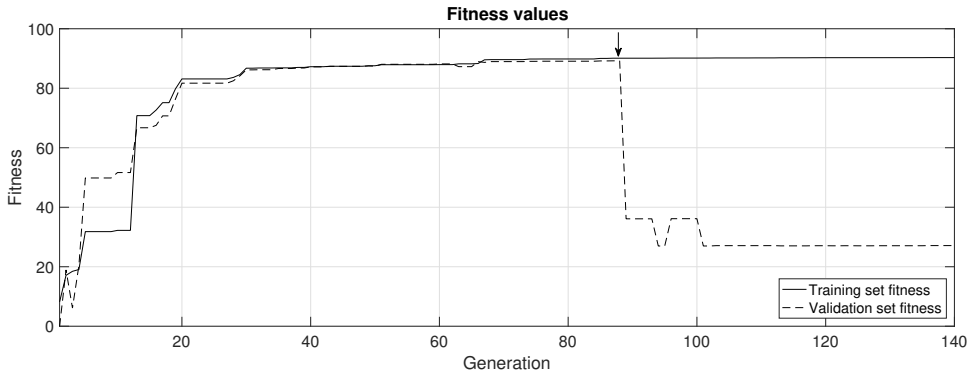

Figure 3: This figure shows the fitness value obtained from the training- and validation set as a function of generations in the GA that were used to optimize a FFNN. The possible fitness values range from 0 to 100. It can clearly be seen that the two fitness values start to diverge at generation 88 marked by the arrow, and most probably this is were the algorithm start to overfit the network to the training set.

The FFNN

10 hidden neurons was found to work best and the noise parameter in the squashing function, c, was set to 1. The initial range of weights were set to [-10,10] to allow the full range of values from the squashing function.

The Test Program

The testprogram found in "TestProgram.m" should run by it self. It loads the best individual which is stored in "bestIndividual.mat", and then performs the simulation of the truck running down the slope specified by the two variables iDataSet and iSlope found at the top of the script. The program makes use of "GetSlopeAngle.m" so to run any of your own test curves please modify this function. Otherwise, the program is the same as the one found in "RunTruck.m" but with the modifications necessary to plot the required data.

The Results

The fitness values corresponding to the training- and validation set can be seen in figure 3. At generation 88 the two values start to diverge where the training fitness increase but the validation fitness decrease, a classic sign of overfitting. The best network was taken as the best one in the 88th generation, indicated by the arrow in the figure. This network managed to complete all the slopes created by the author, which means that a fitness value of about 90 indicates that on average, the truck will go downhill with a speed of $0.9 \cdot 25 = 22.5\,\mathrm{m\,s^{-1}}$. The best obtained network produced fitness values of 90.0952, 89.2140 and 87.9289 for the training set, validation set and test set respectivley.

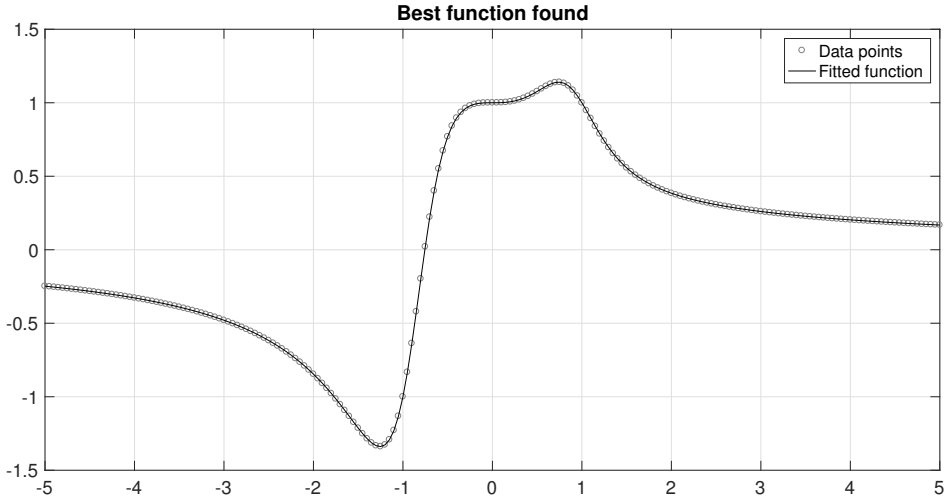

Figure 4: The best function found when fitting the given data points with LGP. The function was found after about 25 000 generations. For more detailed parameter settings, see the text.

Problem 2.4

LGA was used to fit a function to the provided data. A population of 100 individuals was used. The crossover probability was set to 0.8, the mutation rate was set to 5 divided by the chromosome length, and thus varied between chromosomes. Tournament selection was used with selection parameter set to 0.75 and tournament size set to 2. Based on empirical results, the fitness of chromosomes longer than 400 genes were penalized by a multiplicative factor of 400 divided by the number of genes in the particular chromosome. Elitism was used where two copies of the best individual were carried over to the next generation unaltered.
Four operands were used as specified in the problem description, but for the division operator a protected definition was used, where in the case of division of zero, the resulting value was set to 10^{15}.
Four variable registers were used as well as four constants, -1,1,2 and 3.
All the chromosomes were initialized randomly with between 200 and 400 genes. The equation was found to be

$$g(x) = \frac{1 - x^2 + x^3}{1 - x^2 + x^4} \tag{4}$$

which resulted in an error of $2.840 \cdot 10^{-9}$. This particular function was found after about 25 000 generations. It is visualized in figure 4. As expected from evolutionary algorithms the *exact* parameter settings were not crucial for the performance of the algorithm. The values used for this exercise was a combined consideration of typical values given in the text book as well as shorter empirical runs.