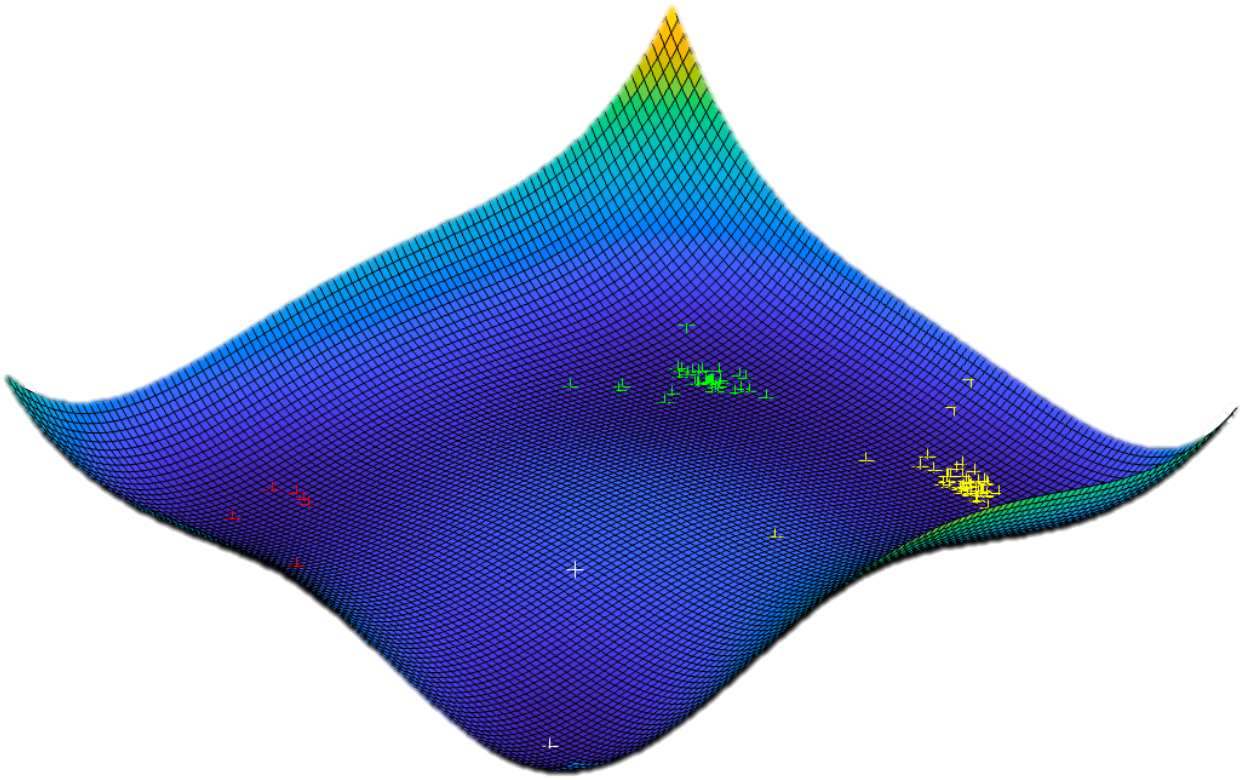

Home problem 2

Contains solutions for:
2.1
2.2
2.3

Written by:
Oscar Rosman
[991215-2791]

Problem 2.1: The traveling salesman problem (TSP)

Result

The shortest path found by the implemented AS algorithm had the length 91.39 length units. This was achieved by reducing the stopping criteria to the shortest path found several times and rerunning the program if it got stuck for too long. When maintaining the normal stopping criteria, the algorithm normally stopped around 98-99 and the jumps were not big enough.

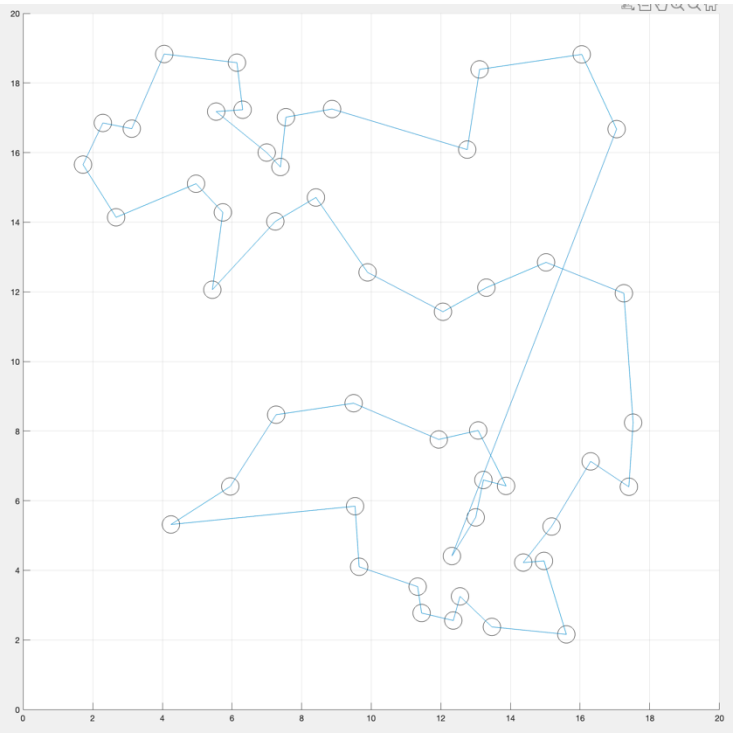

Figure 1) Plot showing the shortest path found by the implemented AS algorithm.

Implementation

Structure (Created functions):

- RunAntSystem.m
  - > InitializePheromoneLevels.m
  - > GetVisibility.m
  - > GeneratePath.m
  - > GetPathLength.m
  - > ComputeDeltaPheromoneLevels.m
  - > UpdatePheromoneLevels.m
- BestResultFound.m

The GeneratePath.m function followed the algorithm closely by implementing a tabu list and calculating the probability for traversing each edge according to the book. One modification was made however, when a city had been visited the visibility to it was set to zero. This resulted in the probability for revisiting this city always being zero, preventing it from being

chosen again. This modification filled the same function as the tabu list but resulted in a lot faster running speeds of the program in general. Because the change was local within the function this did not affect the rest of the program and the visibility matrix outside the function remained unchanged. Because the function was found effective and quite readable with comments no “GetNode.m” function

The other functions in the program were implemented exactly according to the algorithm without deviations.

Problem 2.2: Particle Swarm Optimization (PSO)

Result:

In total 4 minima were identified within the range [-5, 5] for the function f (x_1, x_2). These minima can be seen in the figure below and are more accurately described in the table below it.

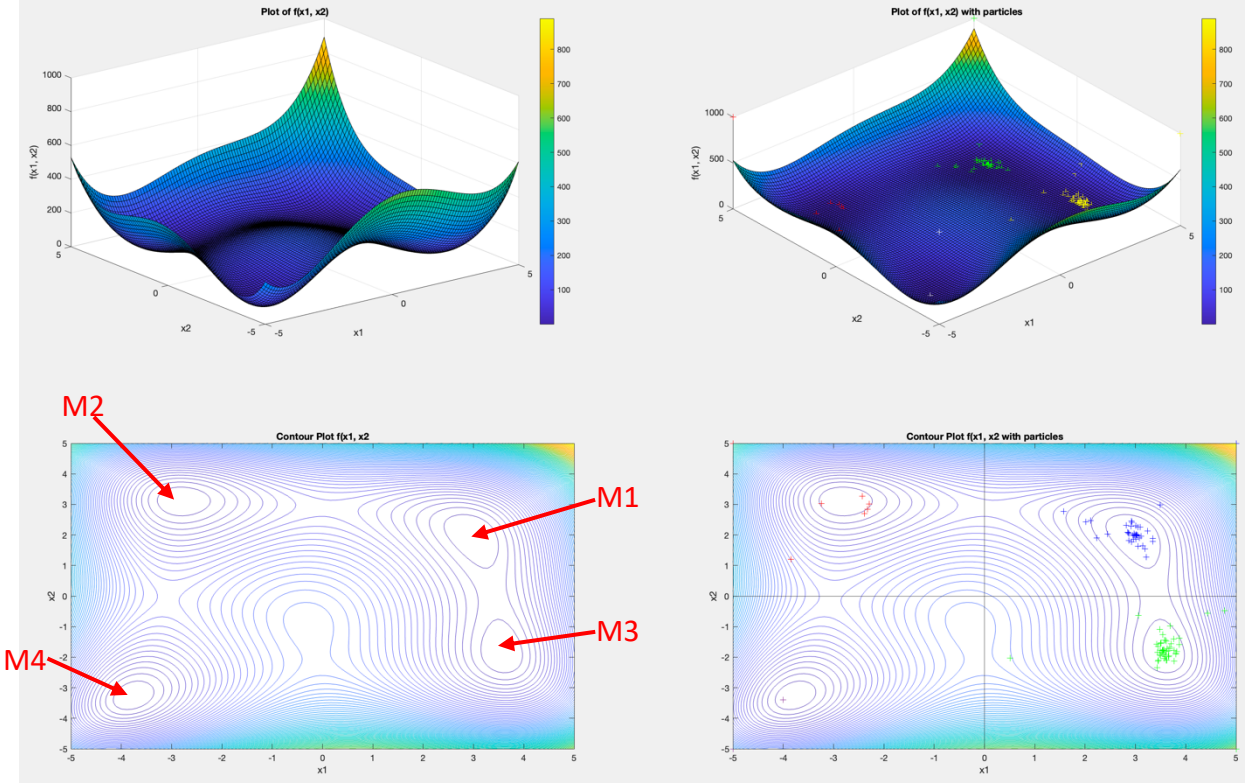

Figure 2) The two upper pictures are plots of the function f within the range [-5, 5] and the bottom two are contours of it. In the two pictures to the right coloured '+' can be seen in three out of the four minima. These '+' each correspond to a particles best position in the run. All minima are marked and indexed with red arrows in the figure in the bottom right.

Minima \ Data	f (x_1, x_2)	x_1	x_2	Colour
Minimum 1	0.00004	2.9989	2.0012	Blue
Minimum 2	0.00001	-2.8051	3.1307	Red
Minimum 3	0.00014	3.5850	-1.8511	Green
Minimum 4	0.06345	-3.7597	-3.3080	Black

Table 1) Shows the lowest point found in each of the minima with function value and coordinates. The last column indicates the colour used for particles near that minimum in the figure in the bottom right.

First, when looking at a plot of the function and a contour plot of it four distinct minima could be identified which were located near the corners [5, 5], [-5, 5], [5, -5] and [-5, 5]. The

figure 1 does not reflect the result in table 1 more than indicating the location of the minima. The data in the table is the result of repeated runs of the constructed PSO.

The program

The problem was solved using a primary program called RunPSO.m which implemented the algorithm described in chapter 5 in the book using external function carrying out specific tasks. The functions used are listed in the bullet points below.

- RunPSO
  - > Initialization.m
  - > Evaluation.m
  - > VelocityUpdate.m
  - > PlotResults.m

During the construction of the program in the majority of runs the particles quickly converged on all minima even if they noticeably struggled with finding the 4^{th} minima located near the corner [-5, -5]. Even if the particles converged closely to the correct minima their best positions could be improved. To solve this an attempt was made to change the parameters and increase the number of particles. What was found to be most effective was converting RunPSO.m to a function and constructing an additional program called RepeatedPSO.m. The new program simply called RunPSO.m 100 times and compared the minima found to the ones previously saved, if a found minimum was better it the old one was overwritten. The results presented were eventually found using this program.

When having identified that four minima existed and their proximity to the corners a very simple k-means inspired classification method was implemented in the function PlotResult.m. This was only for classifying the points to the correct minima after the run and had no effect on the particle swarms actual search for the minima.

Several different values for the parameters were tried within their recommended ranges. For exact values for final selection parameters and the ranges they were chosen from, see the top of RunPSO.m.

Comment:

The handed in files include the program RepeatedPSO.m because it was used and reference, it is not necessary to run the RunPSO for evaluation.

2.3: Optimization of braking systems

Result

The best run was achieved with a network using 10 hidden neurons which achieved a fitness score of 0.48/1.0 on the test set and 0.44/1.0 on the validation set.

Set | Track	Outcome	Avg. speed [m/s]	Fitness	Cause:
Validation:	**4/5**	**21.3**	**0.4426**	
Val. 1	Successful	22.1		
Val. 2	Successful	20.7		
Val. 3	Failed	-		Velocity
Val. 4	Successful	21.4		
Val. 5	Successful	19.2		
Test:	**4/5**	**21.0**	**0.4842**	
Test 1	Successful	22.1		
Test 2	Successful	20.0		
Test 3	Successful	21.0		
Test 4	Failed	-		Velocity
Test 5	Successful	18.8		

Table 2) Result from the best chromosome on the different courses in the validation and test set.

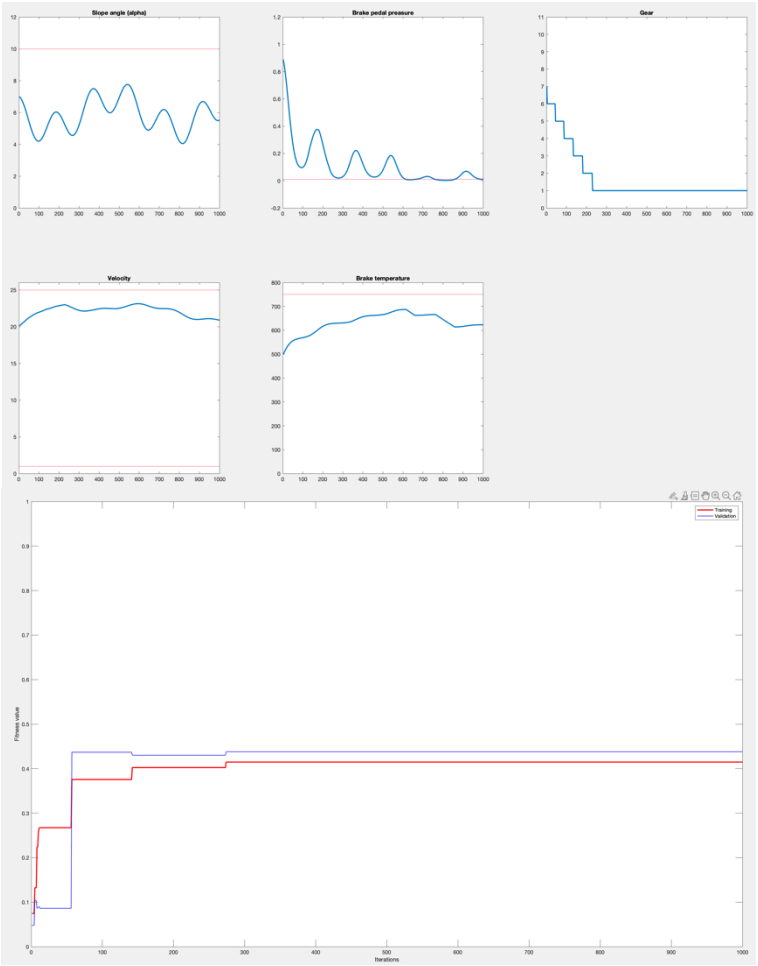

Figure 2) Data from a run with the best chromosome found on the first course in the Validation set.

Figure 3) Fitness values tracked over the iterations in the GA. Blue is the validation set and red is the training set.

Implementation

The program is constructed with a typical GA structure where the program "RunFFNNOptimization.m" implements the GA algorithm with several functions assisting it. The structure is described in the bullet list below.

- RunFFNNOptimization.m
  - > EncodeNetwork.m
  - > DecodeChromosome.m
  - > EvaluateIndividual.m
    - GetSlopeAngle.m
    - TruckModel.m
  - > TournamentSelection.m
  - > Crossover.m
  - > Mutation.m
- RunTest.m
  - > BestChromosome.m

The functions that encode the weights in the network and decodes the chromosomes are quite simple. The encoding function simply flatten the received matrices and joins them together into a single chromosome while reformatting the values between -1 and 1. The decode function reshapes the chromosome into the same two matrices received in the encoding function scaling the values within correctly.

The evaluation function that assigns fitness values to each chromosome tests the network on each track in the training set using the angle received from the "GetSlopeAngle.m" function and the velocity received from the "TruckModel.m" function. If any of the constraints is violated the run on that track is cancelled but accounted for in the fitness value, if the constraints were never violated the model ran for 1000m. The fitness value was then calculated using equation 1 below.

$$(1)\ F_i^{tr} = \frac{\bar{V}}{V_{max}} \times \left(\frac{\sum_{m=1}^{n} x}{1{,}000 \times n} \right)^4, \quad n = number\ of\ tracks$$

The fitness function (eq. 1) takes all tracks into consideration by looking at the average speed and the travelled distance in each track. If the network fails to finish all tracks it is penalised by raising a number smaller than 1 to the power of four which have a large impact, this is more prevalent in the beginning of the GA. Later, when most networks are successful in the majority of tracks a higher average speed gives a larger fitness value. Due to the design of the fitness function a fitness of 1.0 is very difficult to obtain however because it requires the network to maintain a speed of V_{max} the whole track, never going over it and completing all tracks.

Tournament selection with a tournament size of five was chosen for this task to balance exploration and exploitation more easily than RWS would. Crossover and mutation with creep was carried out as in a typical GA.