SOA HW1

Fredrik Sitje
950314-8232

September 2023

Problem 2.1, The travelling salesman problem (TPS)

The goal of this problem was to solve the travelling salesman problem (TSP) by using an Ant Colony Optimisation (ACO) algorithm more specifically an Ant System (AS). The goal is it to find the shortest path in which each city only is visted once except the starting city. In this particular problem there were 50 cities the salesman should visit but each one only once. The number of possible paths is given by $\frac{(n-1)!}{2}$ where n is the number of nodes (cities). It is evident that the size of this problem grows very rapidly as the size of the problem is increased. So for this particular problem there were $\approx 3.04 \times 10^{62}$ possible paths. If one considers for how long the universe exists according to standard believes, one could test 9.0×10^{35} paths per second since the big bang and would still not have tested every route, and thus could not be sure to have found the

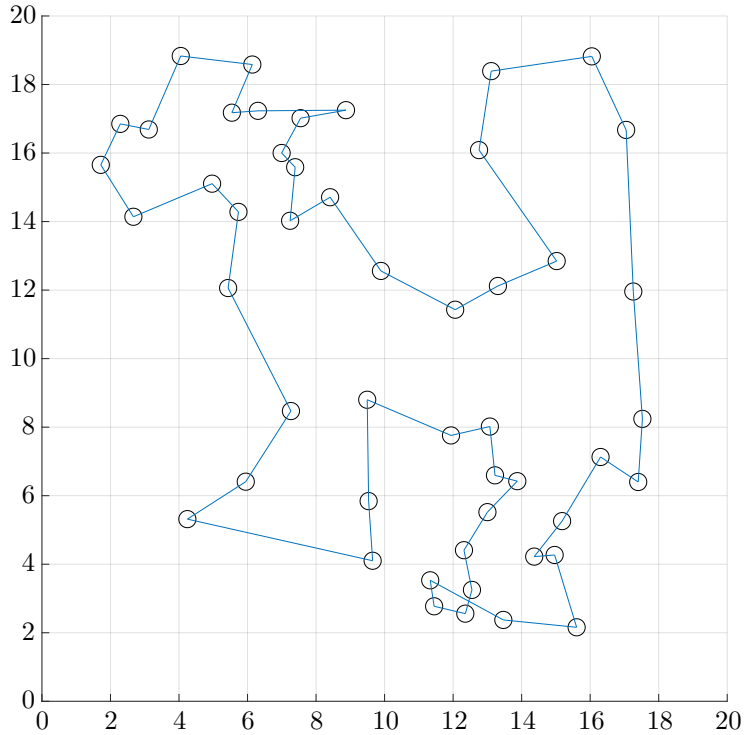

Figure 1: This plot visualises the solution to the Travelling Salesman Problem (TSP) using an Ant Colony Optimization (ACO) algorithm more specific an Ant System (AS) implemented in MATLAB. The TSP involves finding the shortest possible route that visits a set of cities exactly once and returns to the starting city. The cities are indicated by small circles in the figure and the a path is indicated by a line connecting two cities. The key parameters were set accordingly: Number of Ants = 50, Pheromone Importance (α) = 1.0, Distance Importance (β) = 5.0, Pheromone Evaporation Rate (ρ) = 0.5, Initial Pheromone Level (τ_0) = 0.1. After 7 iterations of the ACO algorithm, the best path was discovered by ant number 24, with a length of 98.41967 units. This solution successfully surpassed the target length of 99.99999 units. The connecting lines between the cities represent the path taken by the ant, illustrating the optimised route to solve the TSP.

shortest. Such is the task that the AS can solve easily.

It shall be noted that it did not find the global minima but it found a path below the set target of 99.99999 length units. It is portrayed in figure 1, has the length of 98.41967 units and visited the cities in the following order (represented by the city's index): [25 32 50 29 47 30 34 42 28 46 48 33 38 45 27 39 44 40 37 49 36 8 16 21 7 19 20 11 3 6 15 12 2 24 17 22 14 23 4 10 9 13 1 5 18 26 31 43 35 41].

Different parameters were tested and one could almost see the difference in behaviour when β a constant which regulates the distance importance of one node to the rest of the nodes. The values were chosen in an interval of two to five, while two would set the importance lower the algorithm would give more weight to the paths already discovered by other ants. The contrary would happen for higher values of β as for example for the value of five.

The parameter α in turn regulated the behaviour of the ants in choosing the path depending on the pheromone level of a path. So for high values of α the ant would take the paths which were already used into greater consideration. As one understands, there is an intricate dependence of ant behaviour on the parameters α and β. Considering their influence it might be worth considering to lower the ratio of α/β in the end stadium. This would help the algorithm to focus locally in clusters of cities to sort out the nearest city and choose that path, which might be a good alternative.

One problem the AS has is that it is based on positive reinforcement. This is on the one hand due to the fact that paths that are chosen more frequent have a higher pheromone level and on the other hand that there is an evaporation rate ρ of the pheromone level which can contribute to the reinforcement problem. This becomes especially prominent in the end stadium of the algorithm when one closes in on the global minima, because there is no reason to leave the path with the highest pheromone level, so all paths that aren't at least in a local optimum start to loose importance. The only thing mitigating this is a roulette-wheel selection that bases the selection on stochastics. But the selection is moderated by the ratio of α/β, which leads to the conclusion that it could be recommended to lower the ratio by introducing a dynamic ratio, in case one has clusters of cities and the path does not change much anymore.

Problem 2.2, particle swarm optimisation

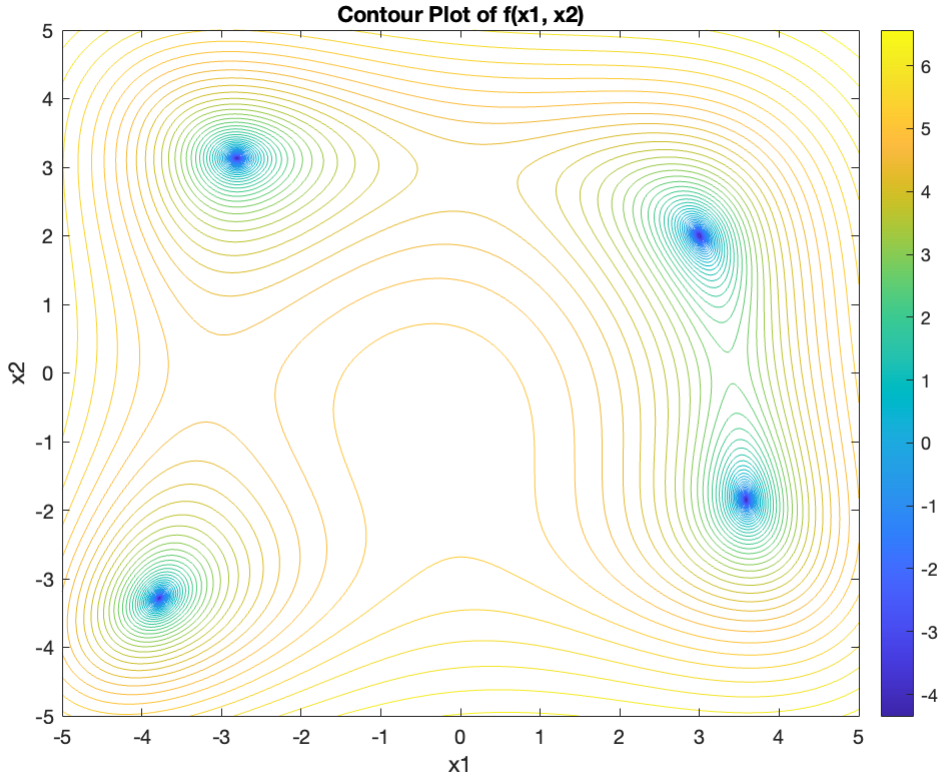

Figure 2: Contour plot illustrating the landscape of the function $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2 - 7)^2$, with local minima identified. The plot is generated over the range $(x_1, x_2) \in [-5, 5]$, and the function values are transformed by adding a small positive constant (0.01) and taking the logarithm for clarity. Local minima are marked as distinct low points on the contour map. The contours reveal the complex of the function, making it an good candidate for optimisation using Particle Swarm Optimisation (PSO).

Table 1: Local minima and corresponding gradient values of the function $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2 - 7)^2$, identified using Particle Swarm Optimization (PSO). The table lists the coordinates (x_1, x_2) of the local minima, along with the partial derivatives $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$ at those points. Additionally, the function values $f(x_1, x_2)$ at each local minimum are presented.

x_1	x_2	$\frac{\partial f}{\partial x_1}$	$\frac{\partial f}{\partial x_2}$	$f(x_1, x_2)$
-3.7793	-3.2832	0.0016	-0.0015	0
-2.8051	3.1313	0.0012	-9.834×10^{-4}	0
3.0000	2.0000	0	0	0
3.5844	-1.8481	-0.0028	5.8105×10^{-4}	0

Problem 2.3, optimisation of braking systems

To solve this problem a feed forward neural network (FFNN) was trained with a genetic algorithm (GA) in Matlab. The goal was to train the FFNN to take control of the breaking behaviour of a truck in artificial alps and thereby let the truck go as fast as possible over a slope of 1000 meters while staying within given constraints. Thus the fitness of an FFNN on a given slope i, was defined as the average velocity times the distance driven, $\bar{v}_i d_i$. Each FFNN was trained on ten different slopes and then validated with five other slopes. At this point it is important to mention that no information about the validation slopes was used to train the chromosomes. It was decided to use the average of the fitness values received to determine the fitness of a chromosome. Further, the method of holdout validation was used to stop the training. This was implemented the following way, if the validation fitness would not grow or stay the same for the coming 2000 generations the training would stop. In figure 3 one can see the training fitness compared with the validation fitness of the best chromosome per generation, and, indicated by a vertical line at generation 1753 of 3753, when the algorithm `RunFFNNOptimization.m` decided to stop the training. This chromosome had an average validation fitness of 23,028 on the validation slopes. The best chromosome could then be further analysed using the `RunTest.m` where a set of five test slopes were loaded to try the overall best chromosome. After a test slope run the the slope angle, the break pedal pressure, the gear status, the velocity and the break temperature were plotted as a function of distance.

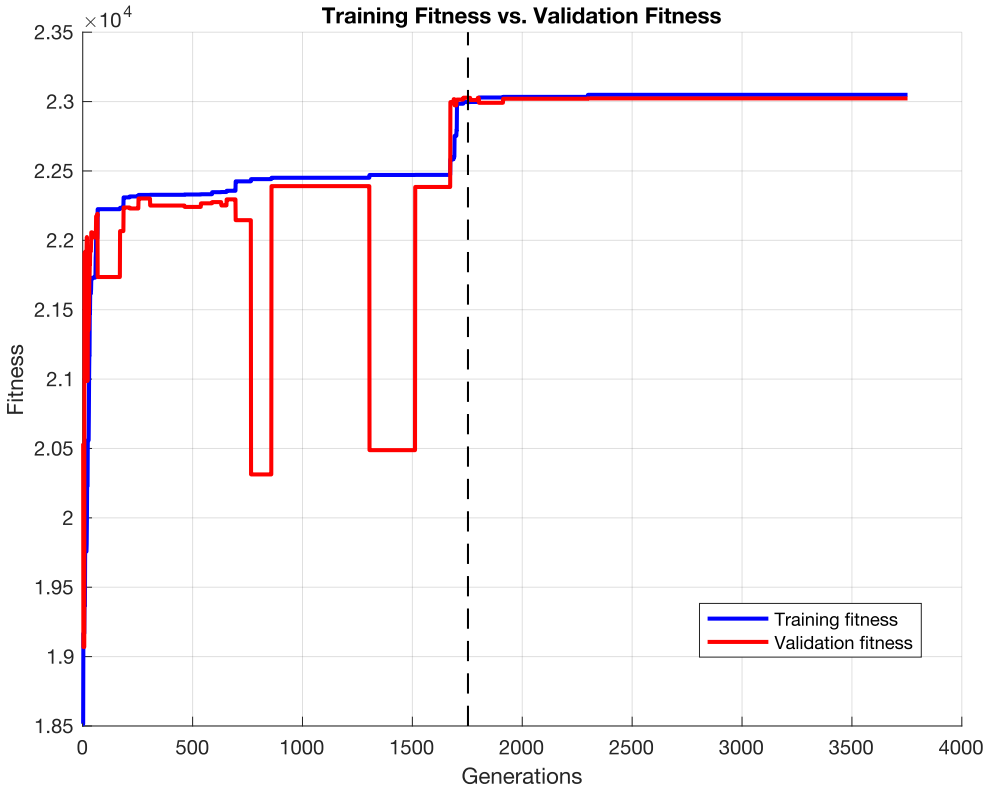

Figure 3: In this figure the training and validation fitness of the best chromosome of each generation is plotted. In total there were 3753 generations and the holdout validation criterion stopped the training and found the best chromosome to be at generation 1753 with a validation fitness value of 23,028.

The FFNN had three inputs, one hidden layer with seven neurons and an output layer with two neurons. The neurons used the a sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-cx}}$$

which turned the signals propagating to the output into values between zero and one. The constant was set to $c = 2$.

To connect such an FFNN to a GA it is necessary to create a chromosome decoder and encoder.

The weights x were encoded to real number genes using the following formula

$$g = \frac{x + d}{2d}.$$

Each of the weights would then placed in a specific order to build a chromosome. This was done with the `EncodeNetwork.m` function. The inverse was done using the `DecodeChromosome.m` function.

The whole `RunFFNNOptimization.m` begins with initialisation of 100 random chromosomes, which each is decoded and fed to a truck simulator called which is essentially a while loop. The constituents which together build the truck simulator are called `GetSlopeAngle.m`, `FNN.m`, `GetTemperature.m`, `TruckModel` and `GearBox.m`. With these functions, some starting values and the input to the FFNN the training was done. The FFNN was essentially trained to regulate the pedal break pressure P_p and the gear which the truck was driving in depending on three input parameters which were the velocity, the angle of the slope and the break temperature T_b. While the breaking pressure could be interpreted from one of the output channels as the signal indicating 0-100%, the gear change had to be decided by dividing the possible output channel signal ([0,1]) into three regions. If the signal was below 0.33 it was requested to change the gear up, between 0.33 and 0.66 to change gear down and for more than 0.66 to keep the current gear. It's deliberately stated as a request, since it was forbidden to change gears more than on time per two seconds. Also it was forbidden to change into a higher or lower gear if current gear was the lowest or highest. The break criterion was defined to be, the truck is not allowed to drive more than 1000 meters, the break temperature must be smaller than 750 Kelvin, the velocity must be between 1 and 25 meters per second and the total time was not allowed to be higher than 200 seconds. This meccano was repeated for 100 chromosomes per generation.

The GA was then implemented with tournament, crossover, mutation and creep mutations. The tournament probability was set to 0.75, the crossover probability to 0.3 (which is fairly low), the mutation probability was set 1 divided by the chromosome length, the creep mutation probability to a high 0.8 with a creep rate of 0.005. The crossover probability was deliberately chosen to be low since in most cases a random cut of weights and mixing does usually not increase the performance of a FFNN. The creep mutation was introduced as an addition since a standard mutation could change a whole weight and that could have dire consequences for the performance of a FFNN. So the creep mutation would change a gene according to

$$g \leftarrow g - \frac{C_r}{2} + C_r r$$

where C_r is the creep rate and r is a random uniform number in [0,1].

It is important to notice `RunFFNNOptimization.m` does the optimisation of the FFNN solely with the help of the evolutionary pressure of growing the average fitness over the five validation slopes. It is, to my surprise, a very pleasant insight to see a different method of training a FFNN than gradient descent.

Problem 2.4, function fitting using LGP

It's was not possible to get any satisfying results in due time, while there is a lot of code in the zip file.