Home Problem 2 Report

Oscar Bark, 950711-5831 CID: baoscar

October 2017

Problem 2.1, The traveling salesman problem (TSP)

a)

To demonstrate how quickly the Traveling salesman problem scales with the number of cities, the number of distinct paths is calculated by first determining the total number of paths, and then excluding those that are equivalent. The total number of possible routes between N cities is simply $N!$, because there are $N!$ permutations of the set $\{n\}_{n=1}^{N}$. But because all routes which share the order of run through cities are equivalent for this problem, it does not matter which city comes first. Therefore, one can divide the number of possible routes by N. Also, since all permutations of two sequences in opposite order are equivalent, the number of routes can be divided by 2. In conclusion, the total number of distinct paths N_p in the Traveling salesman problem is given by equation (1):

$$N_p = \frac{(N-1)!}{2} \tag{1}$$

b)

A program was written in MATLAB to find the shortest path in the TSP using a Genetic Algorithm, starting from a randomly initialized population. The program is simply run by calling GA21b.m. After the run, it prints the path length of the best path found.

c)

Using the Matlab script `AntSystem.m` from the course web page as a starting point, a program implementing Ant Colony Optimization was written to solve the same problem as in **b)**. Just like the previous program, this one is run by calling the main function, `AntSystem.m` once.

d)

The so called nearest neighbor path is a sub-optimal, but good solution to the TSP. To calculate this path length, starting from a randomly sampled city,

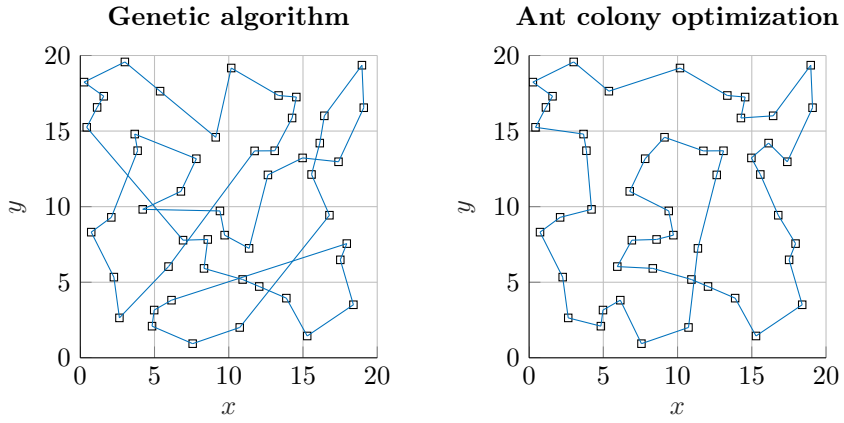

Figure 1: The shortest paths obtained by the GA and the ACO for the Traveling salesman problem. The length of the path found by the GA is 164,94, whereas the path length found by the ACO is 122,62.

a program `NNPathLengthCalculator.m` was written. To demonstrate the advantage ACO has over GA by quickly finding the nearest neighbor path, the expected value of the path over the 50 starting cities was calculated with help of the aforementioned program. This value was found to be 143.5 whereas the shortest path length obtained by a long run, described in **e)**, of the GA implemented in **b)** was found to be 164.94, greater than the nearest neighbor path length.

e)

A long run was performed using each of the optimization methods. The GA was run over 10000 generations with the parameters $N = 200$, $p_{muy} = 0.04$, $p_{tour} = 0.8$ and $k = 2$. The run took approximately 3 minutes on the computer used. The ACO, with parameters $N = 50$, $\alpha = 1$, $\beta = 4$ and $\rho = 0.5$ was run until a solution with path length below 123 was found. This program had a run time in the order of seconds. In both cases, the parameter values specified here are left by default in the program. The shortest lengths of the shortest paths found by the algorithms and the nearest neighbour path, are displayed in table 1.

Table 1: The shortest path lengths obtained with each of the algorithms.

Method	Shortest path length
Nearest neighbour	143,5
Genetic algorithm	164,94
Ant colony optimization	122,62

Conclusively, the ACO was able to find a significantly shorter path than the GA in a shorter time. This is clearly reflected upon inspection of the paths plotted in figure 1. Because the ACO, unlike the GA, was given information regarding a pre-calculated nearest neighbor path in the initialization of the pheromone levels τ, the comparison is not entirely fair. Although the ACO is supposed to be well suited for the TSP, a more fair comparison between the algorithm could perhaps be made by initializing the GA population with the various nearest neighbor paths, or to set the initial pheromone levels $\tau_0 = 1/D$ where D is the average path length of a number of randomly generated paths.
The shortest path found is saved in the file `BestResultFound.m`. Load the path vector by simply calling the script.

Problem 2.2, Particle swarm optimization (PSO)

In order to find the multiple minima of the function given by equation (2), a Particle Swarm Optimization program was written. The program has two settings: One where the swarm best position x^{sb} is defined as the best position in the current iteration, and one where it is defined as the best position found be the swarm during the run. (The best position being the coordinates (x, y) which minimizes $f(x, y)$.) The setting is chosen by modifying line 20 in the main code `PSO.m`. Set `i` to either 1 or 2 in `bestDefinition = bestDefinitions{i}`, where 1 denotes the former stated definition of x^{sb}, and 2 denotes the latter.

$$f(x, y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2 \tag{2}$$

The optimization is run by calling `PSO.m`. After a number of iterations (default 100), the best minimum found is printed in the command window along with the corresponding function value.
The program was run over 100 iterations multiple times with the same settings to find the four minima displayed in figure 2. The settings were as follows: $N = 30$, $\alpha = 1$, $\Delta t = 1$ and $c_1 = c_2 = 1$. The resulting values are displayed in table 2 along with the corresponding function values, all equal to 0. It was noted that setting the swarm best position definition to 'All swarms', yielded greater diversity in the solutions than 'Current swarm' which more often converged to the minimum $(x^*, y^*) = (3, 2)$ than other minima.

Table 2: Table of the minima (x^*, y^*) of $f(x, y)$ given by eq. (2). The values were found using PSO.

$f(x^*, y^*)$	(x^*, y^*)
0	$(3.000022, 2.000015)$
0	$(3.584428, -1.848127)$
0	$(-3.779308, -3.283193)$
0	$(-2.805118, 3.131313)$

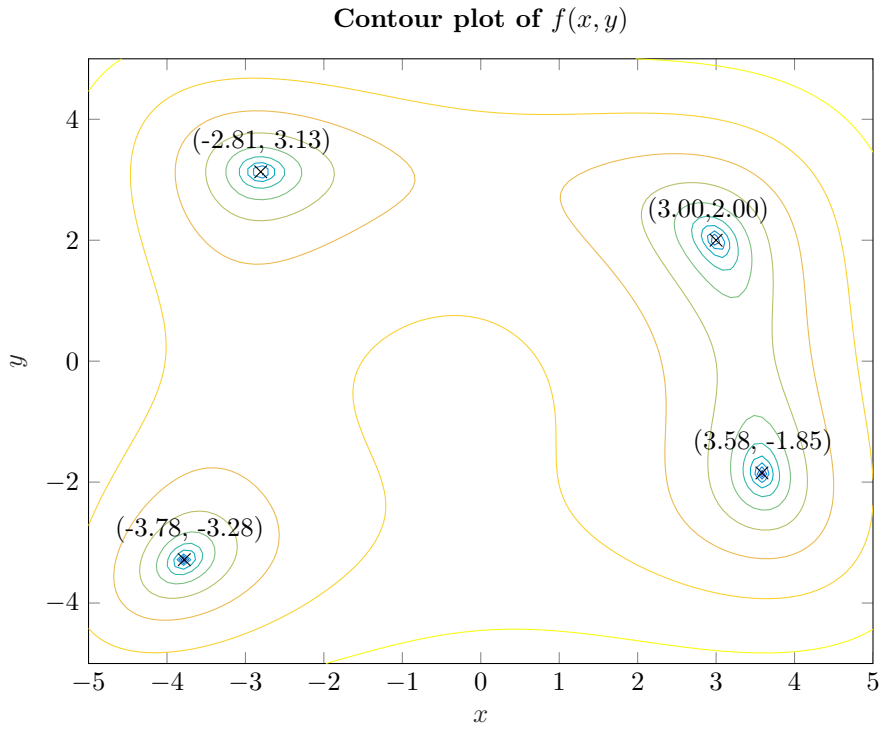

Figure 2: Contour plot of the logarithmically scaled function $\log(0.01 + f(x, y))$ where $f(x, y)$ is the objective function to minizime using PSO. The minima points (displayed in table 2) are marked in the figure by the crosses along with their coordinates found by the PSO algorithm. The values varied slightly between runs.

Problem 2.3, Optimization of braking systems

To control the braking system of a truck running down a downhill slope, a neural network was trained using a GA over 750 generations with parameters $N = 100$, $p_{mut} = 4/m$, $p_{tour} = 0.7$, $k = 2$ and $p_{cross} = 0.3$. The fitness measure of a network controlling the truck on a given slope is defined to be the average speed times the distance traveled across it. The fitness over a set of slopes is taken to be the minimum of the slope fitness scores. The neural network was a feed forward perceptron with one hidden layer of 8 hidden neurons, taking the relative speed v/v_{max}, slope angle α/α_{max} and brake temperature $T_b/T_{b,max}$ as input and outputting the brake pressure P_p and a gear change request Δ_{gear}. If $\Delta_{gear} < 1/3$ or $\Delta_{gear} > 2/3$, the truck shifts gear up or down respectively, assuming its rest period time has passed since the last gear shift.
The truck is implemented as a Matlab handle class `TruckModel.m` in the program, storing the dynamic (speed, horizontal position, brake temperature), controllable (gear, brake pressure) and constant properties of an instance of a truck. Likewise, the controller (neural network) is implemented in a class `FeedForwardNetwork.m` where weight element $w_{ij}^{(1)}$ denotes the directed connection strength from node j of layer 1 to node i of layer 2.
To train the network with the program, run the main file `GA23.m`. The best network found in the run described above can be loaded and evaluated on a given slope by running `TestProgram.m` where `iDataSet` and `iSlope` denote which slope.

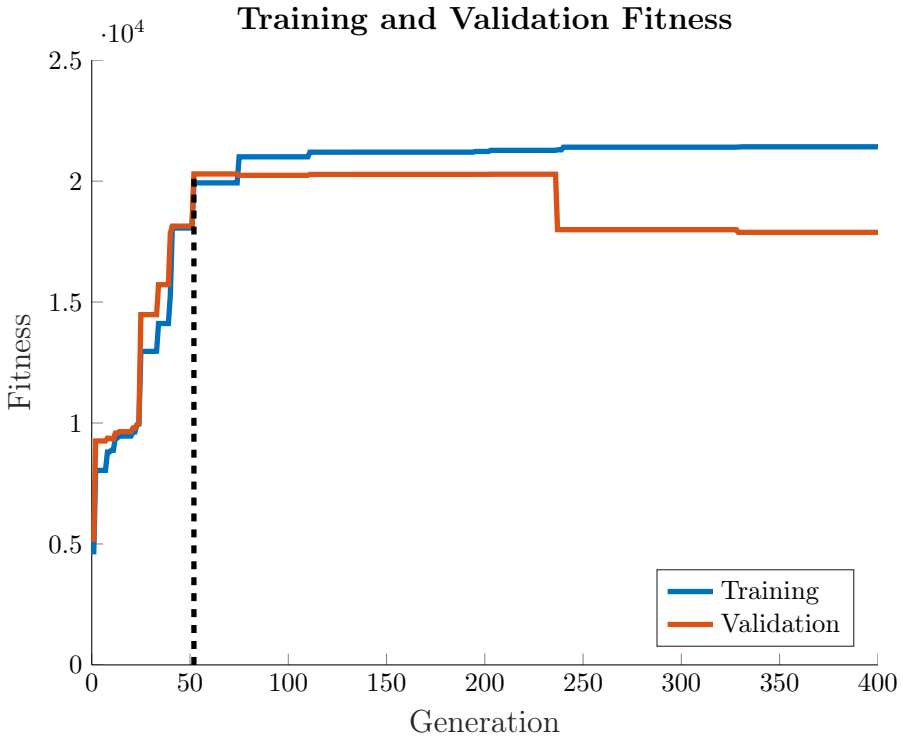

Figure 3: Training (blue) and validation fitness (orange) of the network which performed best on the training set in each generation of a GA. The dashed horizontal line at generation 52 denotes where the network performing best on the validation set is found. The plot shows the results over 400 of the 750 evaluated generations.

Eventuellt skriva något om fitness-utvecklingen över generationerna.

Table 3: Fitness scores achieved by the network trained with the GA on the slopes of the test set.

Slope	$\alpha(x)$	Fitness score ($\bar{v}d$)
1	$6 - \sin\left(\frac{x}{100}\right) - \cos\left(\frac{\sqrt{7}x}{100}\right)$	22699.28
2	$3 + \cos\left(\frac{x}{40}\right) - \sin\left(\frac{\sqrt{3}x}{160}\right)$	21492.99
3	$5.5 + 1.5\sin\left(\frac{\sqrt{20}x}{100}\right) - \cos\left(\frac{x}{80}\right)$	22595.34
4	$1 + \left(\frac{x}{500}\right) + \sin\left(\frac{x}{200}\right)$	21237.68
5	$4 + \left(\frac{x}{1000}\right) + \sin\left(\frac{x}{70}\right) + \cos\left(\frac{\sqrt{7}x}{100}\right)$	21303.43

The resulting network is able to control the truck to achieve fitness scores above

21000 on all five slopes in the test set. The results and the slope functions $\alpha(x)$ are listed in table 3. This likely indicates quite good performance, since the theoretical maximum on a slope is $F_{max} = 25000$, and the truck managed to cover the entire slope of 1000 meters in all cases. Further indication that the network learns some kind of complex behaviour can be seen in figure 4. The controllable state of the truck (brake pedal pressure and gear) seemingly changes to keep the dynamics (speed and brake temperature) within the constraints, applying more pressure on the brake pedals and shifting the gear down when the slope angle increases, vice-versa.

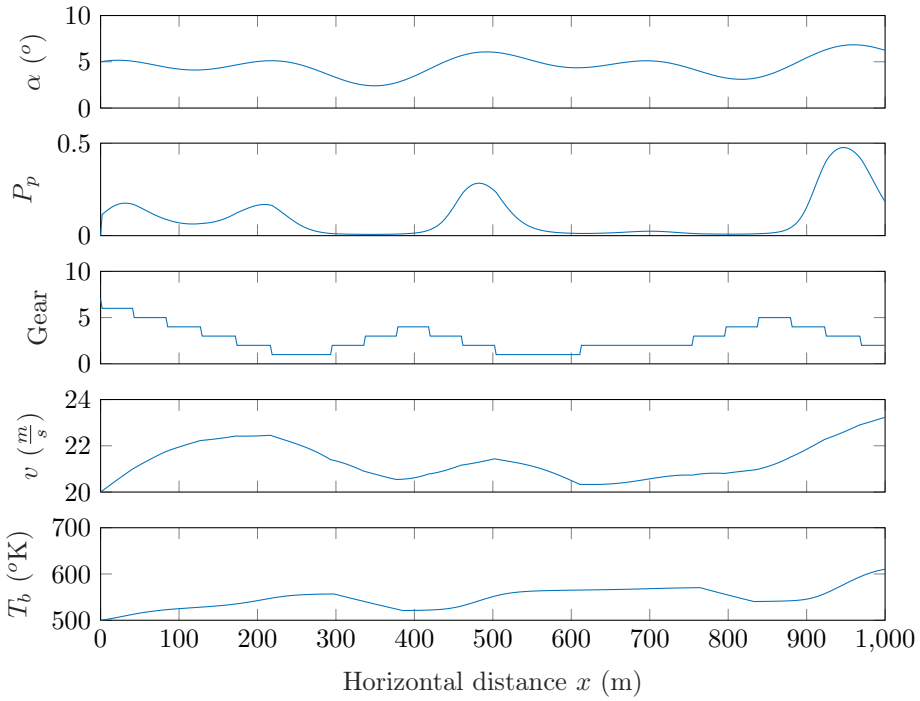

Figure 4: The slope angle and the state of the truck controlled by the neural network trained with the GA during a simulation on slope 5 of the test set as a function of the horizontal distance x traveled.

It is to be noted that the problem is quite sensitive to the data provided. Achieving high fitness scores on slopes that are steeper was found to be significantly harder than doing so on kind slopes, due to the physical nature of the problem. In order to confidently train a network that generalizes well, a set of sloped with high diversity in characteristics (slope angle range, rate of growth, etc.) would be needed, and it is hard to say whether the 20 manually generated slopes provided are sufficient.

Problem 2.4, Function fitting using LGP

To fit an unknown function $g(x)$ to a set of data $\{x, g(x)\}$, a sequence of instructions was generated using LGP with a total of three variable and three constant registers. The constant registers c_1, c_2, c_3 used to find the result were set to $1, 3 - 1$ respectively. A maximum number of instructions (set of 4 genes) was set to 30. Any individual breaking this constraint was penalized in the evaluation by a multiplicative penalty factor exponentially decreasing with the number of instructions above the threshold.
The GA was run over 5000 generations with parameters $N = 100$, $p_{tour} = 0.75$, $k = 5$ and $p_{cross} = 0.2$. The mutation probability p_{mut} varied with the generation count according to $p_{mut} = e^{-\frac{q}{100}} + 0.03$ where q is the number of evaluated generations. (Capped at $p_{mut} = 1$.)
To run the program, call the main script `LGP24.m`. Note that due to the massive search space of the problem, desirable results will unlikely be produced in a few generations.
The best function $\hat{g}(x)$ to approximate $g(x)$ found is specified in equation (3).

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

To evaluate this function, run the script `TestFit.m`, which plots $\hat{g}(x)$ along with the original data, and prints the total RMS error. The total error was found to be $2.847959 \cdot 10^{-9}$. This indicates a good fit, which is clearly reflected in figure 5

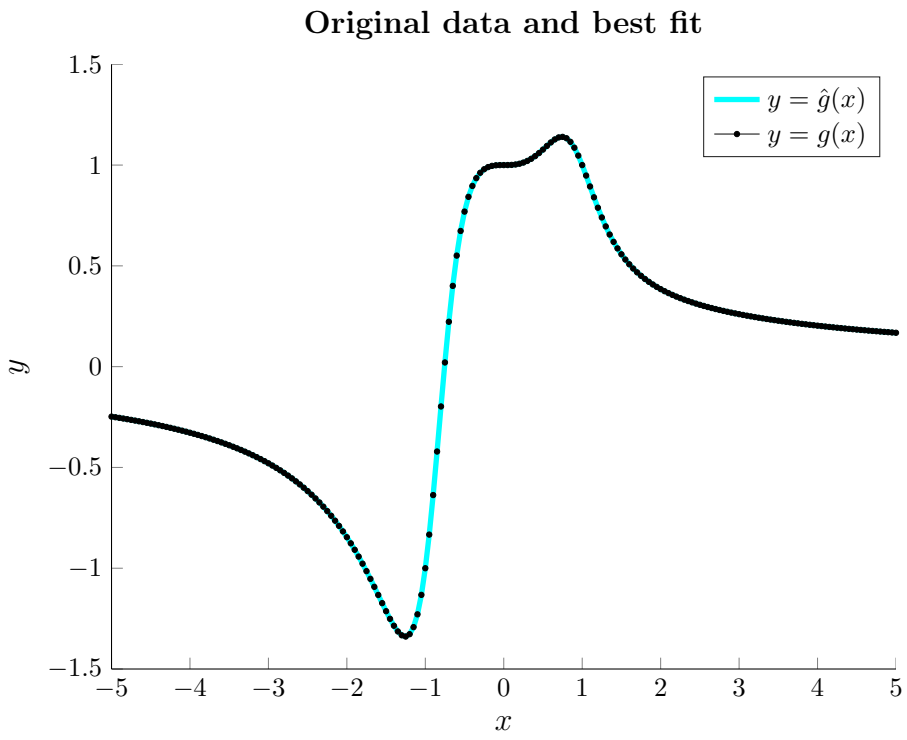

Figure 5: Original data points (black dots) plotted along with the curve of $\hat{g}(x)$ which was fitted to the data using LGP.