CHALMERS UNIVERSITY OF TECHNOLOGY

FFR105 - Stochastic optimization algorithms

David Kastö

950207-8554

Home problem 2

Thursday 18th October, 2018

Problem 2.1 - The traveling salesman problem (TSP)

The aim was to solve the TSP in a case with 50 cities, where the cost function was taken as the length of the total path. For more details about the implemented algorithms, the reader is referred to Wahde[1].

a)

Paths that begin in different cities, but run through them in the same order, are equivalent in the TSP since they have the same total path length. The same goes for paths that go through a given sequence of cities, but in opposite order. To find the number of distinct (i.e. non-equivalent) paths in the general case of N cities, we start off by noticing that the total number of permutations of an "array" with N elements is equal to $N!$. This since one when moving the first element has N alternatives, whereas for the next element there are only $N-1$ options left and for the third $N-2$ etc. Now, for each configuration we know that the opposite order, for instance (1,2,...,$N-1,N$) and (N,$N-1$,...,2,1), yields the same total distance and we should therefore divide the total number of configurations by two. Lastly, we know that we can translate each element in the array by a constant, e.g. move each element two steps to the right, and still get the same total path length. Each element can occupy N different positions in the array and consequently these cyclic permutations render N equivalent array configurations. Thus, the total number of configurations should also be divided by N. All in all, one then obtains that the total number of distinct paths, $n_{distinct}$ can be written as

$$n_{distinct} = \frac{N!}{2N} = \frac{(N-1)!}{2}.$$

However, note that this expression only is valid for $N \geq 3$. For $N = 2$ there is one distinct path (simply the direct path in between the two cities in any direction).

b)

To search for the shortest path between the $N = 50$ cities a genetic algorithm (GA) with permutation encoding was used, i.e. each gene in a chromosome corresponded to a city index between 1 and 50. This list of integers corresponded to the path to be taken in between all the cities. In addition to some permutation of the numbers 1 to 50, a final element equal to the starting index was added to the list during the evaluation of the fitness measure. The latter was taken as the inverse of the total path length (obtained by calculating the Euclidean distance in Cartesian coordinates between each pair of connecting cities).

Tournament selection with size 3 and selection parameter 0.75 was used. Furthermore, swap mutation (with probability $1/N$) and elitism (one copy of the best individual was inserted into the next generation) was implemented, whereas no crossover operator was used. To run the algorithm one only has to run the file GA21b.m. It will then run the GA and continuously update a plot window displaying the currently best (i.e. the shortest) path.

c)

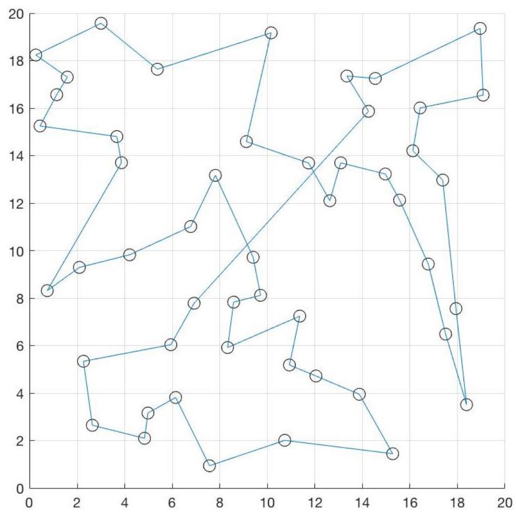
Another approach to solve the TSP described above is to use ant colony optimization (ACO), which is particularly suitable for such routing problems. To this end, an ant system (AS) algorithm, with the main script AntSystem.m, was developed. As in the GA, the currently shortest path found is continuously updated in a displayed plot when the main script is run. The two constants that determined the relative importance for the artificial ants of the pheromone trails and the visibility, the Euclidean distance between two nodes (cities), were set to $\alpha = 1$ and $\beta = 5$. The evaporation rate was set to $\rho = 0.5$.

d)

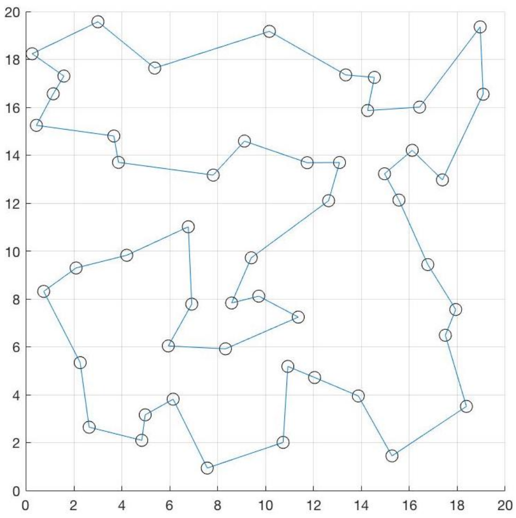
To compare the results acquired by the GA and AS, a script called NNPathLength-Calculator.m was written that calculates (and displays in the prompt) the nearest neighbour path length for a randomly chosen initial city. Running it repeatedly yielded results around 140-160 length units depending on the starting city, e.g. for city index 27 a total nearest neighbour path length of $\sim$ 146.3 was calculated.

e)

Two long runs of the GA and AS scripts created (and stored in the respective subfolders of) in **2.1b)** and **2.1c)** were done. For the GA script, a total path length of $\sim$ 139.73 was found after a run consisting of 100 000 generations. The path is displayed in Figure 1a. The AS script, in turn, managed to find a path with a total path length of $\sim$ 121.60 after 42325 iterations (when the targetPathLength variable in AntSystem.m was changed to 122.0). This path can be seen in Figure 1b.



(a) GA - 139.73



(b) AS - 121.60

Figure 1: Solutions to the given TSP. **(a)** shows the shortest path found, $\sim$ 139.73, by the GA script after a run of 100 000 generations. **(b)** displays the shortest path found, $\sim$ 121.60, by the AS script after 42325 iterations.

In conclusion, this means that both the algorithms managed to get below the total path length of the nearest neighbour path discussed in **2.1d)**. This is not too surprising, since the algorithms are able to consider a more global perspective. In particular, the AS has a global pheromone mapping at every instant and can “plan” the route accordingly, whereas the nearest neighbour approach only has knowledge regarding the distances from a specific city and will only take the local circumstances into account at each step.

The best result (the one obtained by the AS script) was stored as a vector in BestResultFound.m in the subfolder belonging to this subtask, where the last index (denoting the trip back to the city of origin) was excluded. By running this script, the best path (bestPath) is automatically loaded into the workspace.

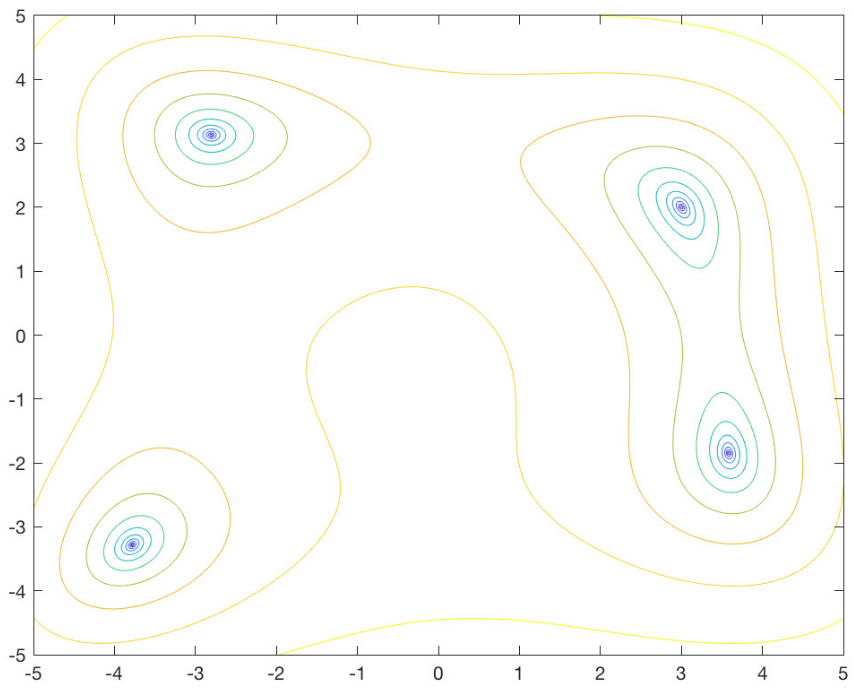


Figure 2: Contour plot of the function $\log(a + f(x,y))$, where $a = 0.01$ and $f(x,y)$ is described by (1).

Problem 2.2 - Particle swarm optimization

The goal was to find the locations of all the local minima of the function

$$f(x,y) = \left(x^2 + y - 11\right)^2 + \left(x + y^2 - 7\right)^2 \tag{1}$$

using particle swarm optimization (PSO). However, first the minima of the function over the range $(x,y) \in [-5,5]$ was searched for with a contour plot. To handle the strong variations of the function value over this range, the contour plot was made for the function $\log(a + f(x,y)$ (with $a = 0.01$), which should have the same optima locations. From the plot, which can be seen in Figure 2, it is clear that $f(x,y)$ have 4 local minima over $[-5,5]$. By using 5000 data points in each direction and zooming in, the respective locations (x,y) were graphically determined to around (-2.81, 3.13), (-3.78, -3.28), (3.0, 2.0), and (3.59, -1.85).

The PSO, with the main script PSO22.m and its wrapper script WrapperForPSO22.m, was used with 30 particles, the time step length $\Delta t = 1$, the velocity constant $\alpha = 1$, the cognitive component $c_1 = 2$ and the social component $c_2 = 2$. The inertia weight w was initialized to 1.4 and then reduced by a factor $\beta = 0.99$ each iteration until the lower limit 0.4 was reached. 1000 iterations were done in total for each run.

By running WrapperForPSO22.m, 150 runs are made and the obtained minima for each run are compared with 8 decimals accuracy. Unique minima and their function values are stored in foundMinima and functionMin respectively. Using the wrapper, four different minima were obtained. Their locations, rounded to six decimals, and function values are listed in Table 1.

x	y	$f(x,y)$
3.584428	-1.848127	0
3.000000	2.000000	0
-2.805118	3.131313	0
-3.779310	-3.283186	0

Table 1: The positions in the x and y direction for the local minima of (1).

Problem 2.3 - Optimization of braking systems

In order to optimize the braking procedure of a truck in a 1000 m long downhill, a genetic algorithm optimizing a neural network was developed in accordance with the description in Wahde[2]. The training is initiated by running the main program

BrakeOptimization.m, which in turn uses TruckModel.m and TruckRunOneSlope.m to train and test the performance. The neural network, defined in NeuralNetwork.m and GetNeuronOutput, is called upon by TruckRunOneSlope.m every time step during a run down a slope. It's GA uses binary encoding with 20 genes per variable, the population size was set to 100, the crossover probability to 0.8, the mutation probability to one divided by the total number of genes, the tournament selection parameter to 0.75, the tournament size to 3 and the variable range to 12.

The fitness measure $F_i = \bar{v}_i d_i$ for slope i was used, where $\bar{v}_i$ denotes the average speed over the evaluation before termination, and d_i the travelled distance before termination. At first the total fitness measure was taken as $F = \min(F_i)$. However, since some of the slopes were basically impossible to go through without violating the constraints regarding maximum speed and/or maximal brake temperature, this seemed to encourage very defensive behaviour, where the truck stuck to the lowest gear even when far below the maximum speed. To mitigate this issue, the total fitness measure was adjusted to $F = \min(F_i) + \bar{F}$, where $\bar{F}$ is the average fitness over the slopes in a set. This change was meant to also award gear increments and speeds closer to the maximum on easier parts of a slope and seemed to have the desired effect.

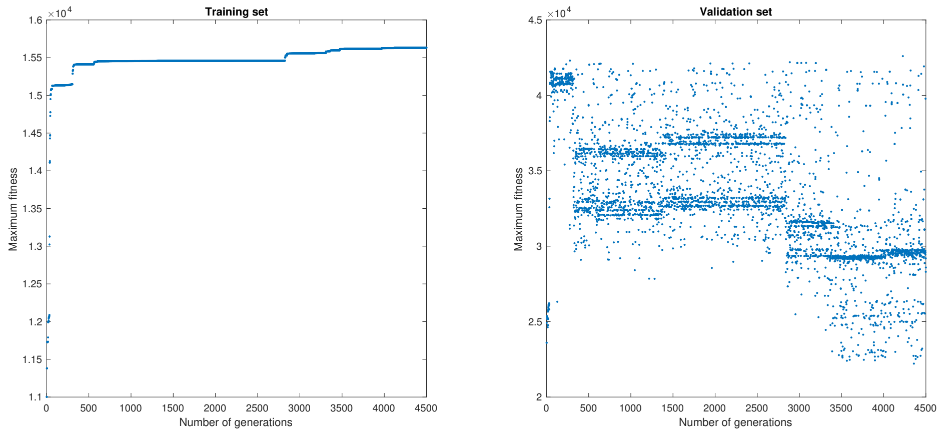


Figure 3: The fitness values for the training and validation sets the first 4500 generations of the best run.

BrakeOptimization has a variable called maxConsecutiveValidationDecreases, with which the training is interrupted when the maximum fitness value for the validation set has not improved for the specified number of consecutive generations and stores the weight that was used when the highest validation fitness value was achieved. For the training session with the highest obtained validation fitness scores, displayed in

Figure 3, 7 hidden neurons were used and the absolutely highest validation fitness was achieved at generation 4227. However, as seen in the figure (treating the more sparse areas as outliers), there is a downward trend for the validation set of the more populated parts from early generations. This, in combination with the fact that generation 278 performed around 1.3% better than 4227 on the test set, led to that the weights of the 278:th generation were chosen as the final weight set.

To try the chosen weights on any given slope defined in GetSlopeAngle, the script TestProgram.m may be used. This program automatically loads and runs the best network on a given slope (e.g. the first slope in the validation set) or an array of slopes, e.g. iDataSet = 3 (denoting the test set) and iSlope = 1:5. When the run is finished, the test program generates a set of subplots (in one plot window per given slope) showing the slope angle, the brake pedal pressure, the gear, the speed, and the brake temperature, as functions of the horizontal distance travelled.

Problem 2.4 - Function fitting using LGP

A linear genetic programming (LGP) program was created in order to fit an unknown function. The main script, LGP24.m, uses a variable register with the length three (where the first element is used to store x) and the constant register [1, 3, -1]. The program had the following parameters; population size 100, maximal chromosome length 180, tournament selection parameter 0.75, tournament size 5 and 1 copy of the best individual to the next generation. The initial range of the number of instructions was 3 to 22. Since it was tricky to get the solution to converge, varying mutation and crossover rates were used. The initial crossover (mutation) probability was 0.4 (0.4) and was multiplied by a factor 0.99 (0.99) every 20th (5th) generation until the final probability 0.2 (1 divided by the maximal chromosome length 180) was reached. If a better solution was not encountered for 1000 generations, the mutation rate was multiplied by 5 (with the same decay rule as above) every thousand generation until a higher maximal fitness was found.

To enforce the maximal chromosome length, a penalty was included. In order to increase the penalty for bigger violations, the difference between the chromosome length and the allowed maximum was taken and then divided by the allowed maximum. Thereafter this value was subtracted from one and then multiplied with the fitness value.

The obtained solution with the smallest error (~ 0.031) was the hefty expression

$$\begin{aligned} y \approx\ & (0.126x^9 - 0.547x^8 + 1.776x^7 - 3.224x^6 + 4.481x^5 - 3.153x^4 \\ & + 0.857x^3 + 2.027x^2 - 1.916x + 1.000)/(0.147x^{10} - 0.483x^9 + 1.261x^8 \\ & - 1.413x^7 + 1.076x^6 + 0.429x^5 - 0.206x^4 - 0.952x^3 + 2.598x^2 \\ & - 2.021x + 1.000). \end{aligned}$$

The curve fit and the original function are displayed in Figure 4.

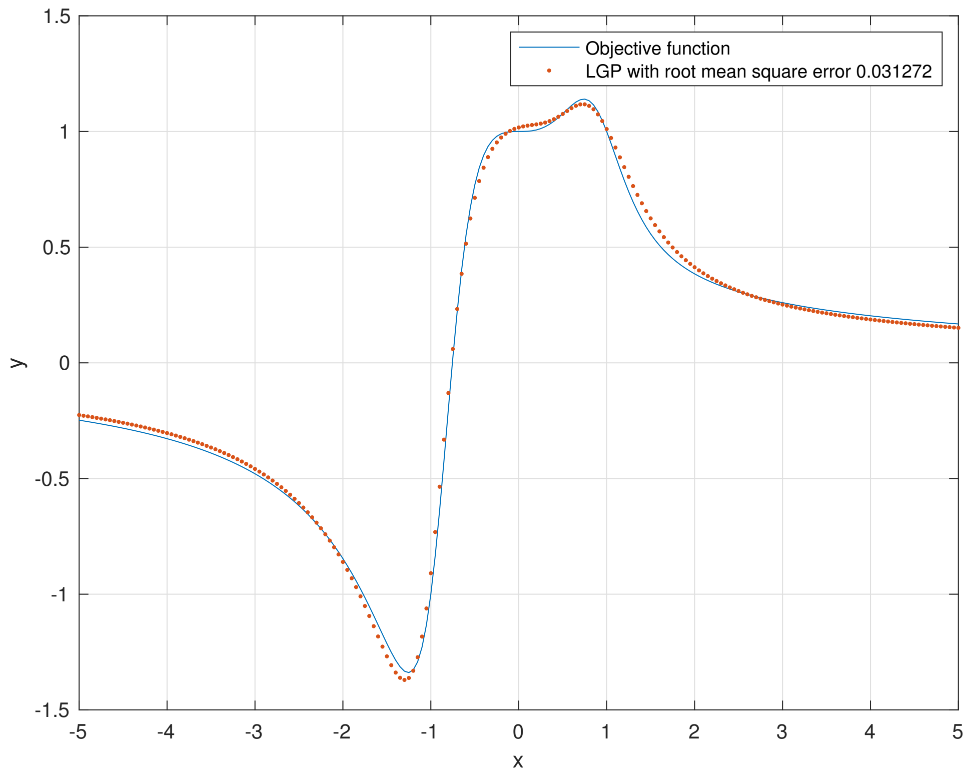


Figure 4: The original function curve and the closest curve fit achieved with the LGP program.

By running TestFit.m, both the original data and the best curve fit are plotted and the root mean square error is written in the prompt.

References

[1] Wahde, M., *Biologically Inspired Optimization Methods: An Introduction.* WIT Press, 2008.

[2] Wahde, M., *Stochastic optimization algorithms 2018 - Home problems, set 2.*