Stochastic optimization algorithms 2018
Home problems, set 2

Cassard Sebastien 19960526-T313

October 18, 2018

1 Problem 2.1 : The traveling salesman problem (TSP)

1.1 Number of distinct paths for N cities

For N cities, the total number of possible paths is given by

$$P_{total} = N!$$

In our case, we only look for *distinct* paths, we need to remove the paths that are equivalent.

We know that paths that go through a given sequence of cities in opposite order are equivalent. As there is 2 possible orders for each path, we need to divide P_{total} by 2.

We also know that paths that start in different cities but run through the cities in the same order are equivalent. Since there are N possible start city, we need to divide P_{total} by N.

Finally, the number of *distinct* path for N cities is given by :

$$P_{distinct} = \frac{P_{total}}{2N} = \frac{N!}{2N} = \frac{(N-1)!}{2}$$

1.2 GA algorithm for TSP

By running the algorithm in the *GA21b.m* file, we obtain the following result:

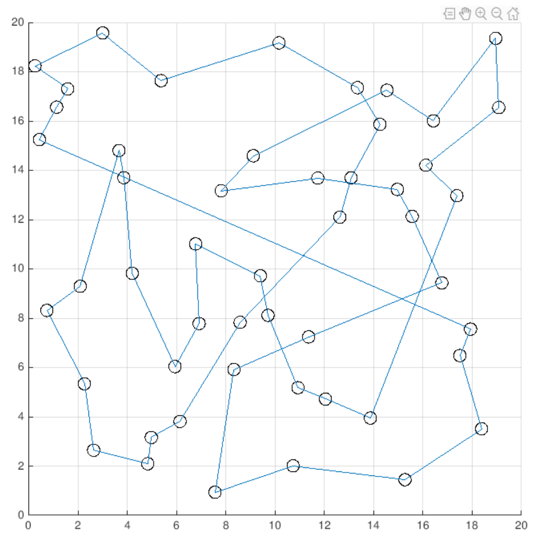

Figure 1: Path obtained with the GA algorithm.

With the following parameters :

- populationSize = 2000
- numberOfGenerations = 100
- mutationProbability = 1/nCities
- tournamentSelectionParameter = 0.75
- tournamentSize = populationSize
- numberOfBestIndToInsert = 1+fix(rand*2)

The path found in the figure above has a length of 171.8625, and corresponds to the following sequence of cities:

$27 \to 15 \to 9 \to 1 \to 5 \to 4 \to 2 \to 47 \to 46 \to 48 \to 40 \to 28 \to 20 \to 22 \to 30 \to 44 \to 41 \to 39 \to 31 \to 21 \to 24 \to 38 \to 43 \to 49 \to 50 \to 42 \to 45 \to 36 \to 32 \to 29 \to 26 \to 25 \to 18 \to 19 \to 16 \to 12 \to 11 \to 10 \to 6 \to 3 \to 7 \to 8 \to 13 \to 14 \to 17 \to 23 \to 33 \to 34 \to 37 \to 35 \to 27$

1.3 ACO algorithm for TSP

By running the algorithm in the *AntSystem.m* file, we obtain the following result:

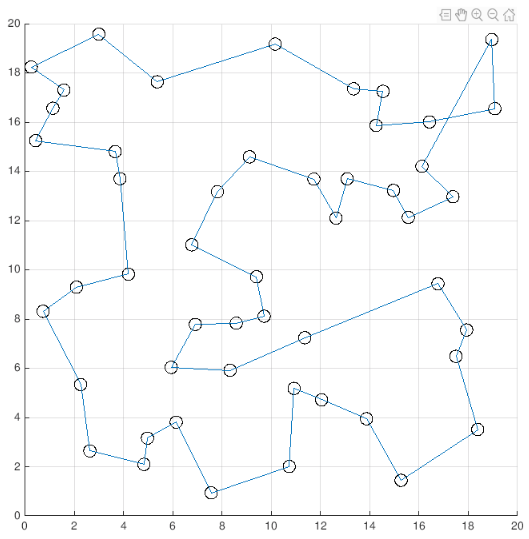

Figure 2: Path obtained with the ACO algorithm.

With the following parameters :

- numberOfAnts = 50
- alpha = 1.0
- beta = 3.0
- rho = 0.5
- nearestNeighbourPathLength = GetNearestNeighbourPathLength(cityLocation)
- tau0 = numberOfAnts/nearestNeighbourPathLength
- targetPathLength = 123.0

The path found in the figure above has a length of 122.33278, and corresponds to the following sequence of cities:

36 → 32 → 29 → 28 → 20 → 17 → 14 → 13 → 8 → 7 → 3 → 6 → 12 → 11 → 10 → 2 → 4 → 5 → 1 → 9 → 15 → 27 → 35 → 38 → 37 → 43 → 50 → 49 → 42 → 45 → 41 → 39 → 34 → 33 → 31 → 24 → 21 → 18 → 25 → 26 → 23 → 19 → 16 → 22 → 30 → 44 → 47 → 46 → 48 → 40 → 36

1.4 NN path algorithm for TSP

By running *NNPathLengthCalculator.m*, we obtained a path length of 162.2709 starting from the city 7. Here is the corresponding path plot :

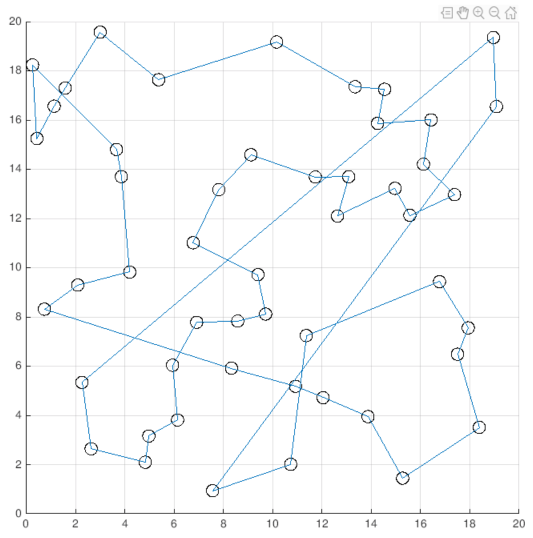

Figure 3: Path obtained with the NN algorithm, starting from city 7.

The length of the path obtained with this algorithm is better than the one obtained by the GA (162.2709 vs 171.8625). An improved version of the previous GA can thus be created, by initializing the population using the nearest neighbour algorithm.

This improved GA was is implemented in *GA_NNstart21.m* and give much better results:

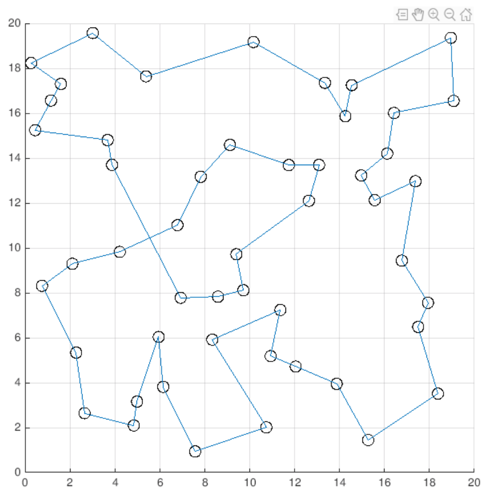

Figure 4: Path obtained with the improved GA algorithm.

The length of the previous path is 126.3693. It can be said that initialization using the NN algorithms drastically improves the performance of the GA, going from a minimum distance of 171.8625 to 126.3693.

The path above corresponds to the following city sequence:

50 → 43 → 42 → 39 → 41 → 45 → 44 → 47 → 46 → 48 → 40 → 36 → 32 → 29 → 30 → 22 → 28 → 20 → 17 → 16 → 14 → 13 → 8 → 7 → 3 → 6 → 12 → 18 → 21 → 24 → 31 → 34 → 33 → 25 → 26 → 23 → 19 → 11 → 10 → 2 → 4 → 5 → 1 → 9 → 15 → 27 → 35 → 37 → 38 → 49 → 50

1.5 Best paths found

Here is a summary of the shortest path lengths obtained with the previous algorithms:

- basic GA algorithm : 171.8625
- NN algorithm : 162.2709
- improved GA (NN initialization) : 126.3693
- ACO algorithm : 122.33278

The best result is the one obtained with the ACO algorithm. The best path obtained is saved in *BestResultFound.m*, and is represented on the following graph:

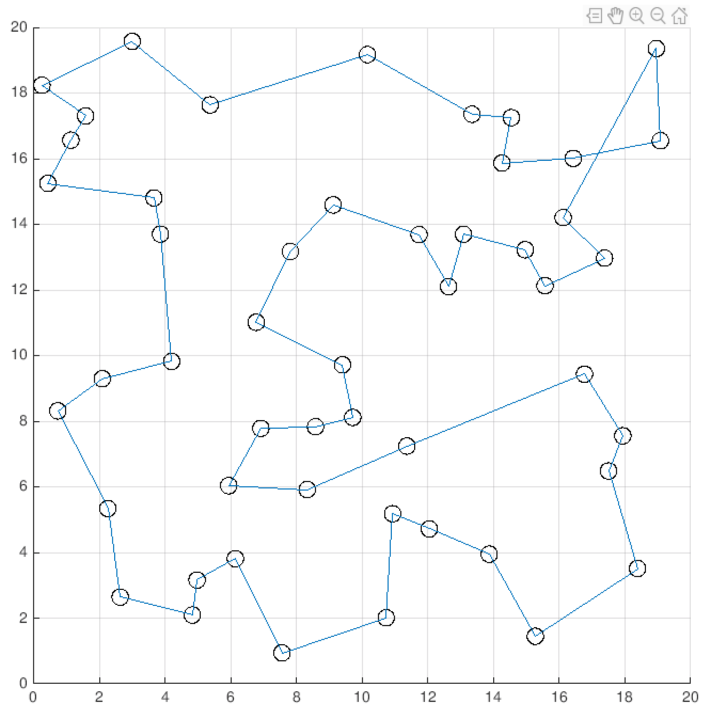

Figure 5: Best Path found (length = 122.33278), obtained with the ACO algorithm.

2 Problem 2.2: Particle swarm optimization

2.1 Contour plot

By displaying the contour plot of $log(0.01 + f(x, y))$, we can get a rough idea of the values of the f minima on $[-5, 5]$.

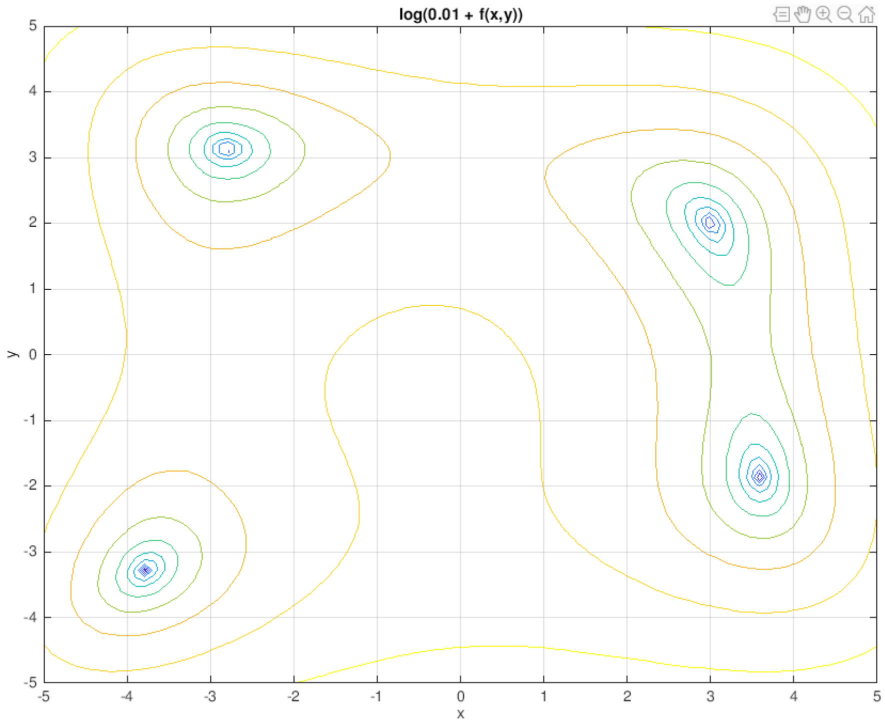

Figure 6:

According to the contour plot above, we find that the minimas are approximately at the following coordinates:

- $(-3.2, 3.1)$
- $(3, 2)$
- $(-3.8, -3.2)$
- $(3.6, -1.9)$

2.2 PSO results

Through different runs, the PSO algorithm found 4 minimums that are coherent with the previous estimations. The following table presents these results:

x	y	f(x,y)
-2.805118	3.131313	9.6347×10^{-12}
3.000000	2.000000	2.7315×10^{-22}
-3.779310	-3.283186	3.7979×10^{-12}
3.584428	-1.848127	1.5320×10^{-24}

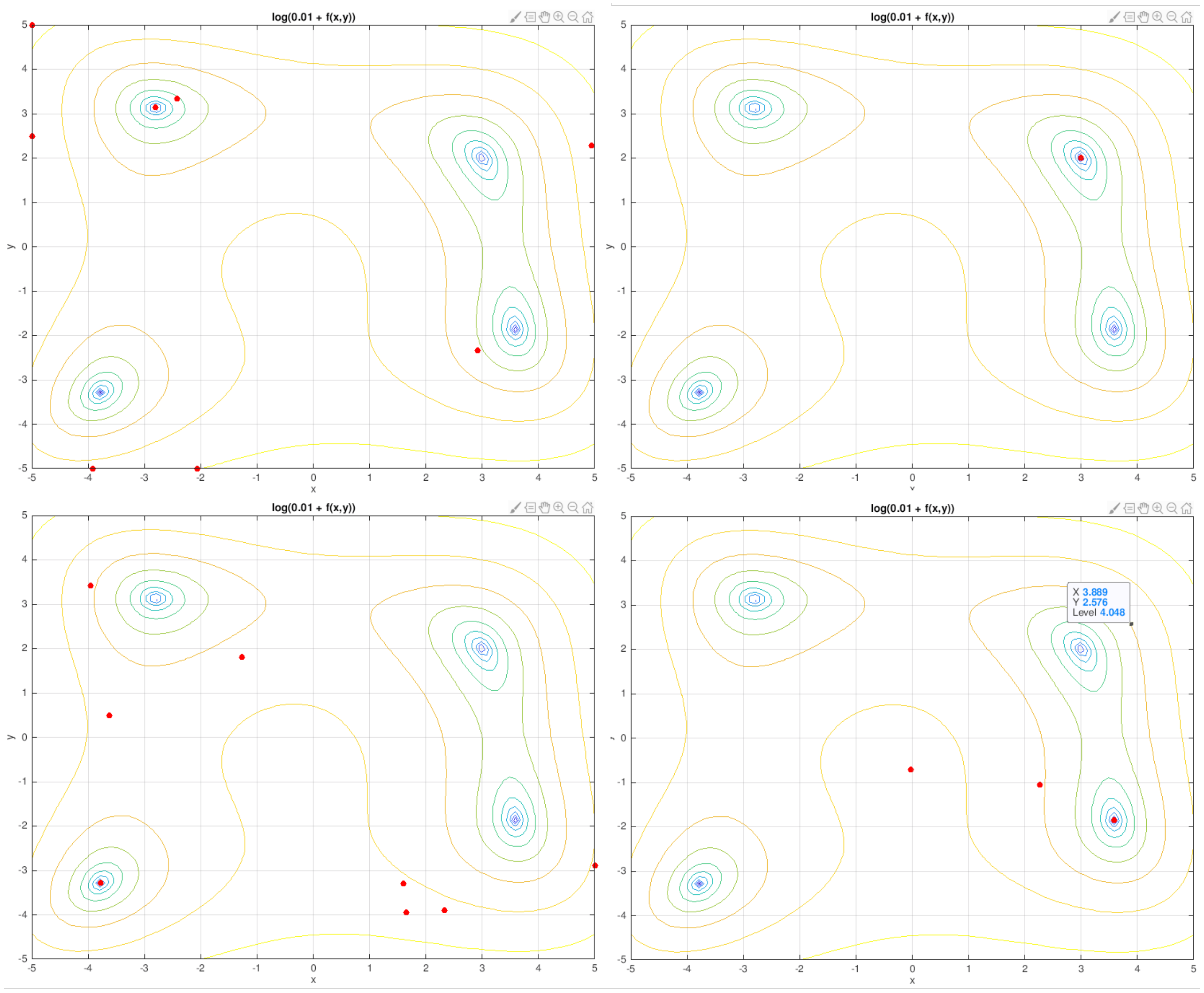

Figure 7: Graphical representations of the minima found

3 Problem 2.4: Particle swarm optimization

Unfortunately, I didn't have time to do this problem (only the Truck Model is implemented).

4 Problem 2.4: Particle swarm optimization

The results presented below were obtained by running the *LGP24.m* program with the following parameters:

- populationSize = 5000
- numberOfGenerations = 500
- nbOperators = 4
- nbVariablesReg = 3
- constantReg = [1]
- nbConstantsReg = size(constantReg, 2)
- eps = 10^{-9}
- minChromosomeLength = 15
- maxChromosomeLength = 150
- crossoverProbability = 0.8
- mutationCoefficient = 5
- tournamentSelectionParameter = 0.75
- tournamentSize = 2
- numberOfBestIndToInsert = 3

4.1 A few words about the parameters

Through various tests, I quickly realized that we had to try to limit the number of registers. Thus, three variable registers and one constant register were chosen. Limiting the number of registers reduces the number of possible instructions, and thus allows the program to find relevant instructions more quickly.

The number of best individuals to insert in each new generation was increased to 3 because it seemed to improve the results of the algorithm, allowing more relevant instructions to be kept between each generation.

The mutation probability was changed to a mutation coefficient C decreasing linearly to 1 during the execution of the program. The probability of mutation is given by $\frac{C}{L}$ where L represents the length of the considered chromosome.

4.2 Results

The execution of LGP algorithm outputs a chromosome with a size of 19 instructions, corresponding to the function :

$$((((((x-(0/x))-(0/x))+((x-(0/x))-(0/x)))+1)/1)*(1/((x+1)+(((x-(0/x))-(0/x))*(x/1)))))$$

This expression could be obtained using the DisplayChromosomeFunction function. A more elegant form of this function is as follows:

$$\frac{2x+1}{x^2+x+1}$$

All workspace variables after the run of the algorithm have been saved and can be accessed by loading the file *bestFunctionFound-Workspace.mat* in Matlab (Warning ! At the time the algorithm was run, the *eps* variable did not exist. It is necessary to define $eps = \frac{1}{c}Max$ to successfully execute the program from this file. Otherwise, the program can be executed without any problem by running *LGP24.m*).

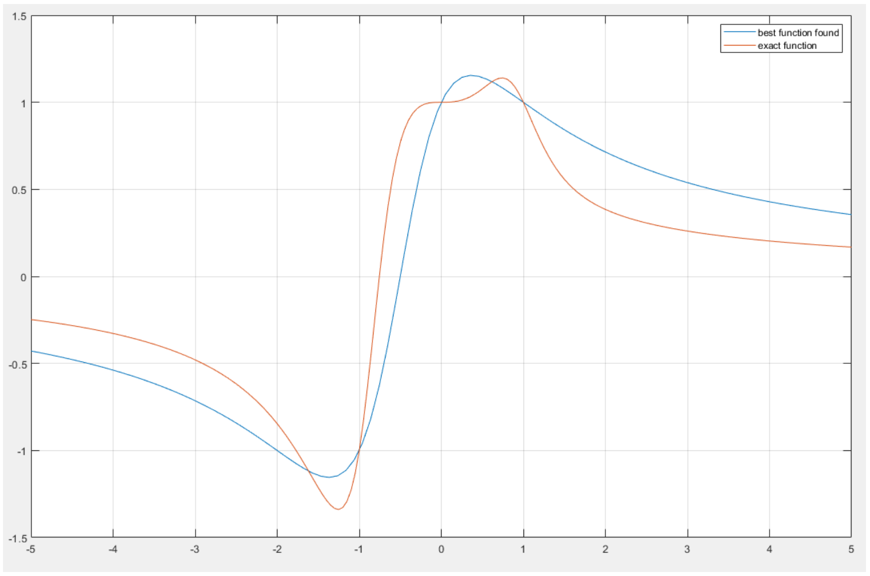

Figure 8: Graph obtained with the *TestFit.m* program.

The RMS error of this result is 0.26533, with a mean error of 0.22718. More details numbers can be found in the *bestFunctionFound.output* file. It would most likely be possible to obtain even more accurate results by running the algorithm over a larger number of individuals and generations. However, the execution time of the algorithm being very long, I didn't have time to start (and finish) a new run of this program...