Natural Computing – Assignment 1 Report

Problem 1 – Analysis of Particle Swarm Optimisation

To find the parameters α1, α2, and ω that perform well, in a short time, under the standard PSO algorithm, I ran the algorithm with different values for the forementioned parameters. For this experiment the following assumptions were made:

- α1 and α2 were set to α1 = α2
- α is set to α1 + α2
- α is capped at 5.00
- ω is capped at 1.00
- Search space dimension *d* is set to 3

The termination criterion is based on three things:

1. Proximity to the global optimum. Since we want good results, and we know that the global optimum is 0, we can stop the algorithm when the best swarm performance is below 0.0001.
2. Swarm Divergence. If divergence occurs the current best fitness is final.
3. Short time. Given that we want to find out good parameter values for a short amount of time the algorithm only runs for 1000 timesteps after initialisation.

It is worth remembering that this is a minimization task, therefore we consider the best values to be the ones closer or equal to 0.

The graph for the **Sphere function** is displayed below.

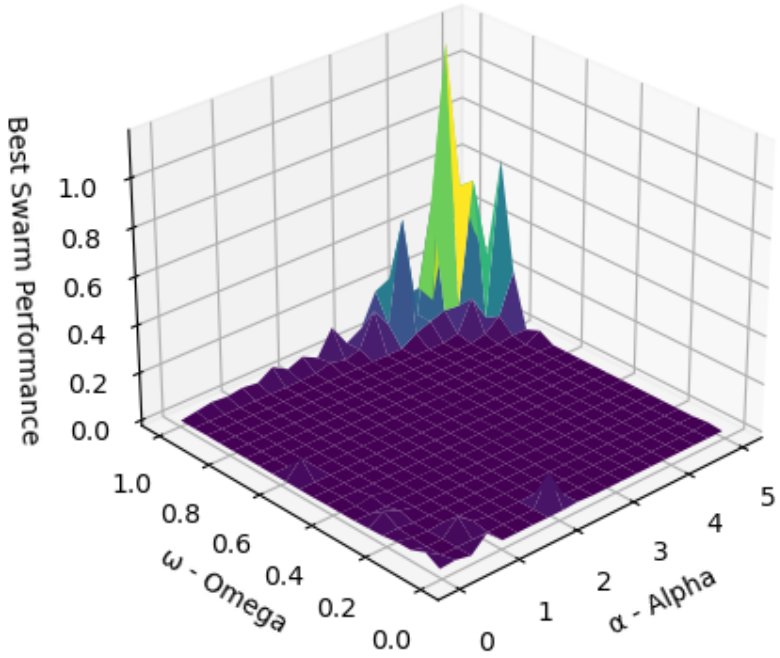

Figure 1: Graph showing the Best Swarm Performance for the Sphere function using the PSO algorithm under different values of α and ω.

As shown in the graph above there is a vast area where the best swarm performance (BSP) is under 0.01 (area coloured in dark purple). High values of α and ω tend to give worse results than lower values. Any values under 3 for α and under 0.8 for ω show really good

results. Despite the peaks at high parameter values of α their value is still under 1, so the overall BSP is really low.

For the **Rastrigin function** the same assumptions were made. The surface produced is displayed below, in two different angles.

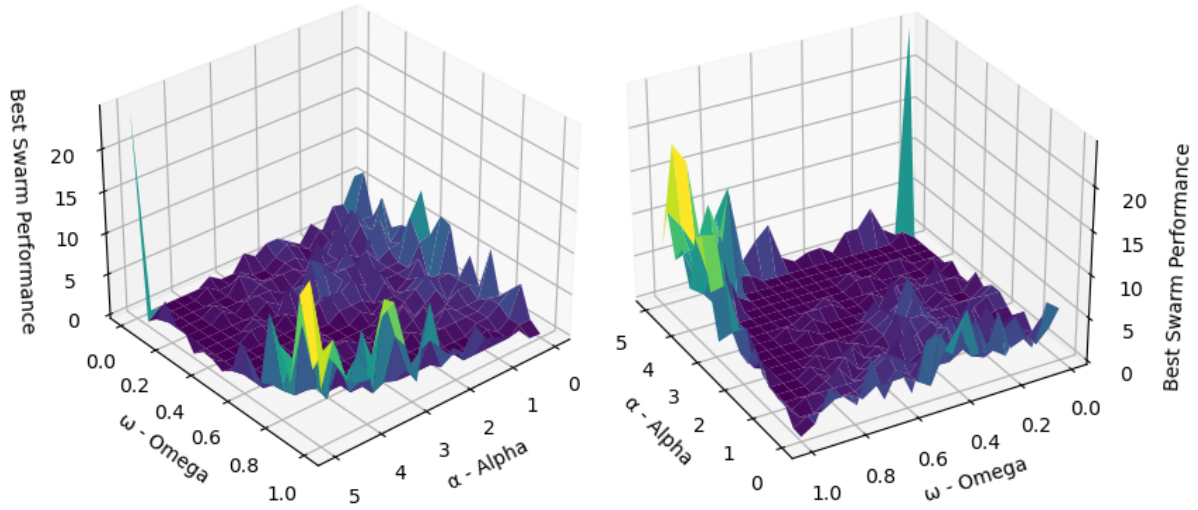

Figure 2: Graph showing Best Swarm Performance for the Rastrigin function using the PSO algorithm under different values of α and ω.

In this case, the area where the BSP is under 0.1 is much lower. Finding a combination of values for these parameters that performs well is more difficult than before, due to the large number of peaks raising up to 5. There are several areas showing extremely good performance. The overall BSP for the parameter search space is not as good as for the Sphere function. Here the BSP raises above 20, whereas it barely surpasses the value of 1 for the Sphere function.

In conclusion, **there is a range of values for α, and ω that results in a good performance** given the stated assumptions. Both exploratory and exploitative parameters are able to find good results for these functions when the space dimension is set to three.

The Sphere function has a good performance under these values:

- $0.01 < \alpha < 3.00$ and $0.01 < \omega < 0.85$

The Rastrigin function has a good performance under these values:

- $1.00 < \alpha < 4.00$ and $0.80 < \omega < 0.95$
- $3.00 < \alpha < 4.50$ and $0.01 < \omega < 0.80$

Problem 2 – Scaling

The parameter chosen for analysis is the search space dimensions *d*. Using the same assumptions as in Problem 1, I ran both functions on different values of *d*. The parameters for both functions were chosen based on the results on Problem 1. For both functions α = 1.50, and ω = 0.85.

The way in which the performance of these functions is evaluated on different search space dimensions is through the Best Swarm Performance (BSP). Below are the results

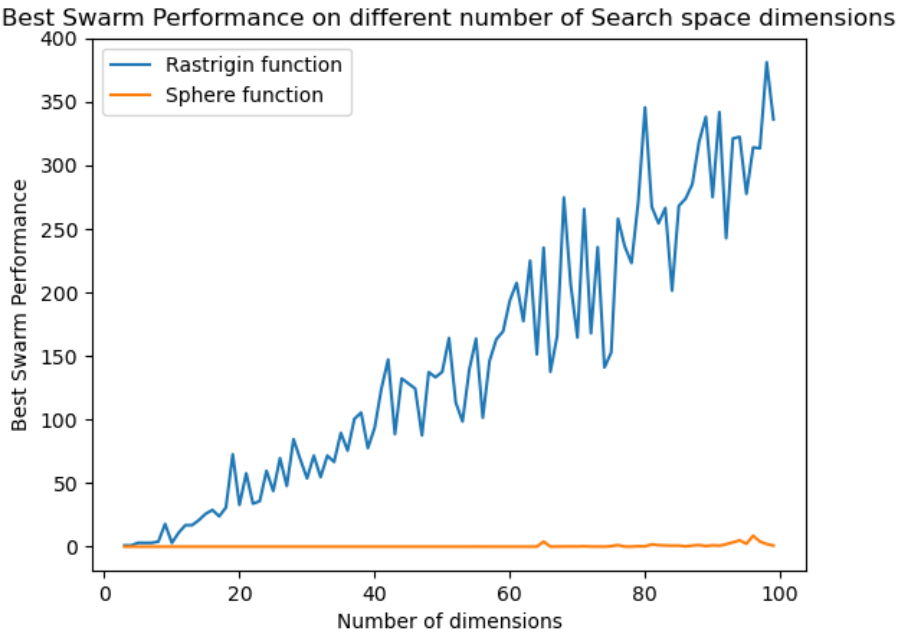

Figure 3: Graph showing the performance of the Rastrigin and Sphere function against different number of dimensions using the PSO algorithm

As seen in the graph, the Sphere function is not dependant on *d*. The flat orange line shows that its BSP does not increase with the number of dimensions. It is only at high values of *d* that the BSP does not reach the global optimum, but it does stay at low values.

The Rastrigin function is dependent on *d*. The blue line shows a clear relationship between the BSP and the number of search space dimensions: as the *d* increases the BSP increases.

The difference in dependence on *d* is most likely due to the **complexity of the functions**. The sphere function behaves in a way where there are no local minima, as values decrease further away from 0 the sphere value increases, the same occurs as you increase away from 0. So, no matter from which direction you approach 0 you will not encounter a local minimum, only the global optimum 0. This does not occur in the Rastrigin function, due to the cos operator the function encounters several local minima. For this reason, as you increase the dimensions there is a higher chance that one of the search space values gets stuck in a local minimum. These two effects can be seen in Figure 4 below.

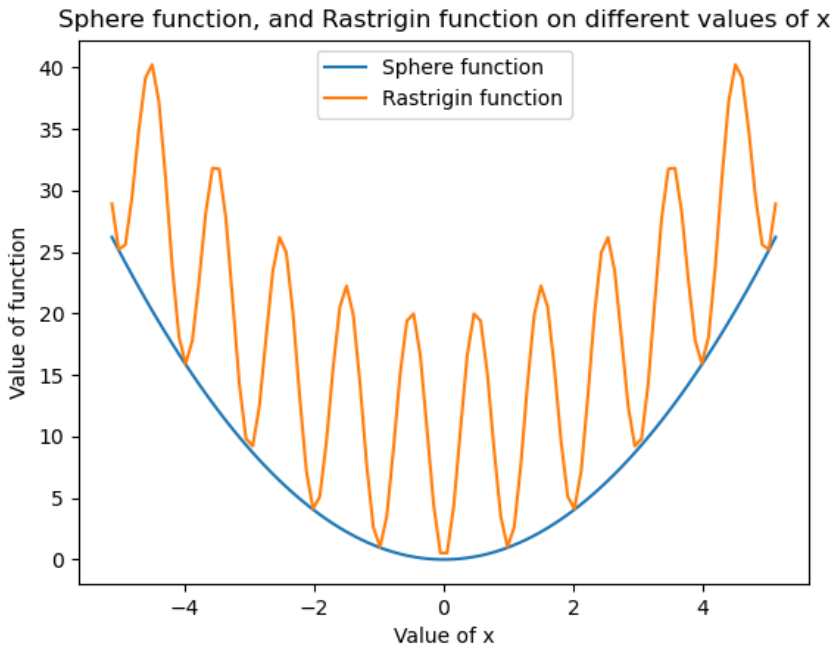

Figure 4: Values of Rastrigin and Sphere function on different values of x (their parameter)

Problem 3 – Heterogeneous particle swarms

To investigate the effect of Heterogeneous particle swarms compared to single-type particle swarms over the Rastrigin function, I compared two different single-typed particle swarms and their heterogeneous combination.
This comparison is made over different values of search space dimensions *d* for the following reasons:

- For low dimensions good parameters will, most likely, result on the BSP dropping to the global optimum, despite the type of particle swarm.
- Figure 3 shows signs of heteroskedasticity, as *d* increases, so does the variance of BSP. For this reason, comparing PSO algorithms on one point is not adequate, as the variance could show misleading results.
- Calculating the performance over different dimensions serves as a strong statistical validation of the results, because the experiment is repeated 100 times with one parameter (*d*) changing at a constant pace.

The first comparison was made under parameter values that reached global optimum in the analysis made in Problem 1. Graph shown below.

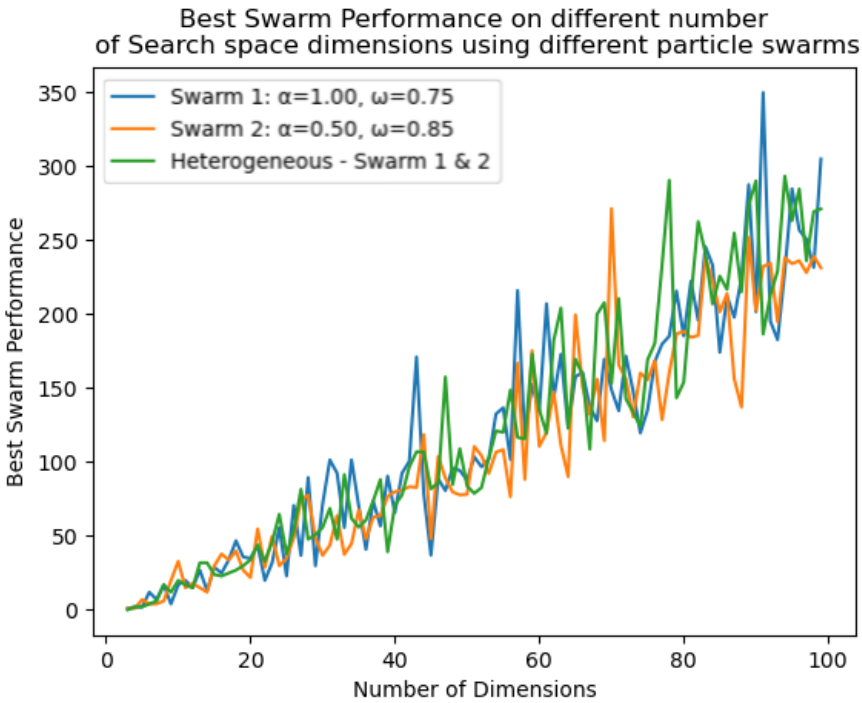

Figure 5: Graph showing the performance of 2 canonical PSO algorithms, with similar parameter values, and a heterogeneous PSO using a combination of the two canonical PSO swarms

The three algorithms show akin results, this is probably caused by the similarity in parameter values. The three variations of the PSO favour exploitation and exploration with the same force (because $\alpha 1 + \alpha 2 = \alpha$), and the α values for both swarms aren't that different, the same occurs for ω.

Calculations on the area under each line show that Swarm 2 performs best. This is, however, by very small margin, probably not significant. The calculations for the area under each line were done using two different rules:

	Trapezoidal Rule	Simpson's Rule
Swarm 1 area	11306	11135
Swarm 2 area	10426	10421
Heterogeneous area	11602	11590

To continue with this analysis the same experiment was carried out using disperse parameter values, including the selection of different values for α1 and α2.

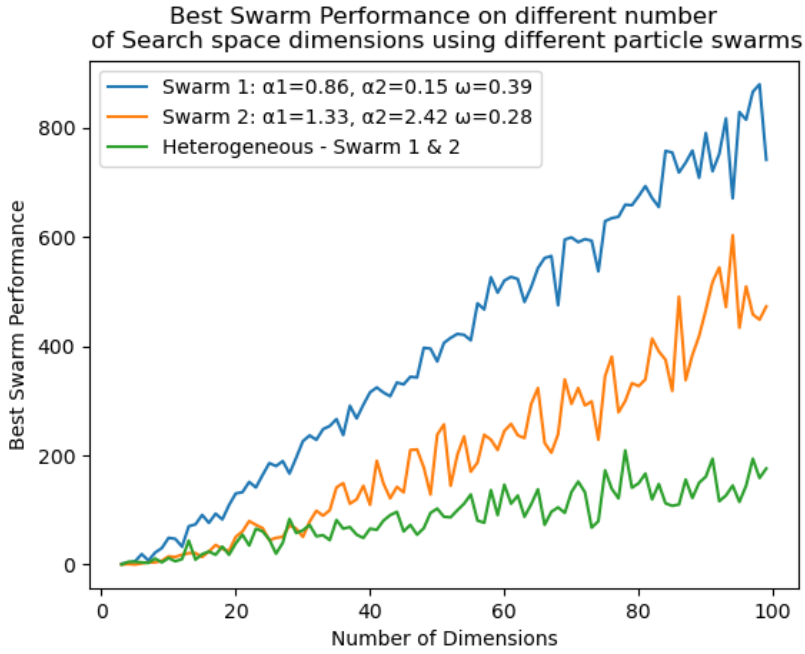

Figure 6: Graph showing the performance of 2 canonical PSO algorithms, with dispersed parameter values, and a heterogeneous PSO using a combination of the two canonical PSO swarms

In this case, there is clear distinction between the swarms. The **heterogeneous swarm shows a better performance** over different search space dimensions compared to the two constituent types. It is also clear that this heterogeneous mix performs better than the one displayed before with optimal parameter values. This can be seen from the clear decrease in the area below the heterogeneous lines (approximately 28.5%). This could be due to the mixture of particles that have more attraction for its personal best and less for the swarm's best and the opposite.

	Trapezoidal Rule	Simpson's Rule
Heterogeneous 1 area	11602	11590
Heterogeneous 2 area	8294	8272

From this investigation we can conclude that a heterogeneous swarm with disperse parameter values, which affords variety in the behaviour of particles, can produce better results, and have higher scalability than single-type PSO.

Problem 4 – Differential evolution

After implementing the Differential Evolution (DE) algorithm, the parameters were set to allow a fair comparison with the PSO algorithm. The parameters that occur in both algorithms (population size, number of iterations, and *d*) were set to the same values as PSO. These are the values:

- Population size = 30
- Maximum number of iterations = 1001
- Crossover rate = 0.7
- Search space dimensions *d* = 3

The problem's specification sets the bound to be between -5.12 and 5-12, so the only free parameter is the scale factor for mutation *F*. To make an appropriate comparison between DE's and PSO's performance on the Rastrigin function, I will find the optimal parameter values, in this case for *F*.

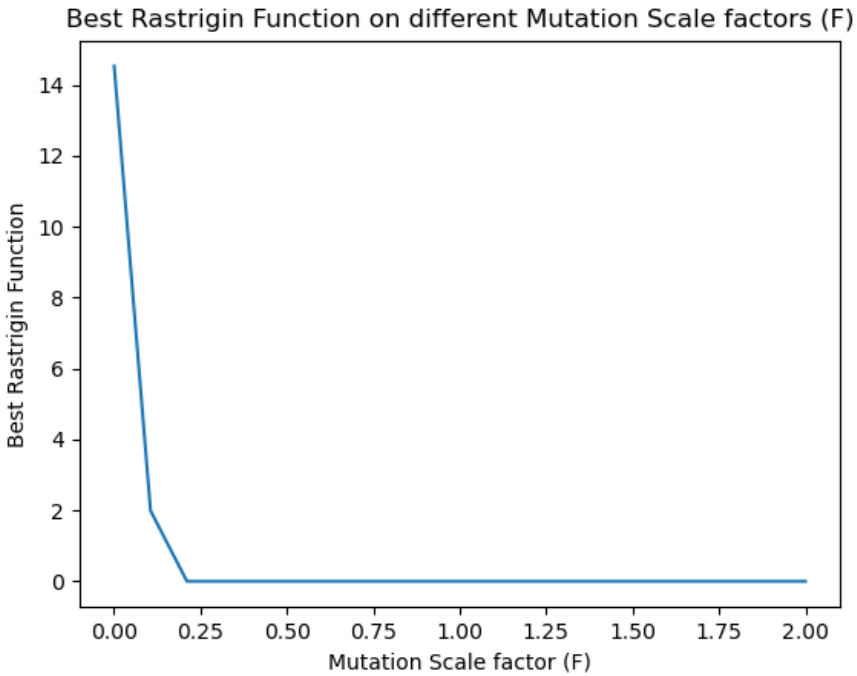

Figure 7: Performance of the Rastrigin function using the Differential Evolution algorithm on different values of the Mutation scale factor

From Figure 7 we can see that the best *F* values are those above 0.25. To start our comparison with PSO, we can now compare the scalability of DE and PSO with their optimised parameter values. The legend of the graph below shows the parameter values used for each algorithm.

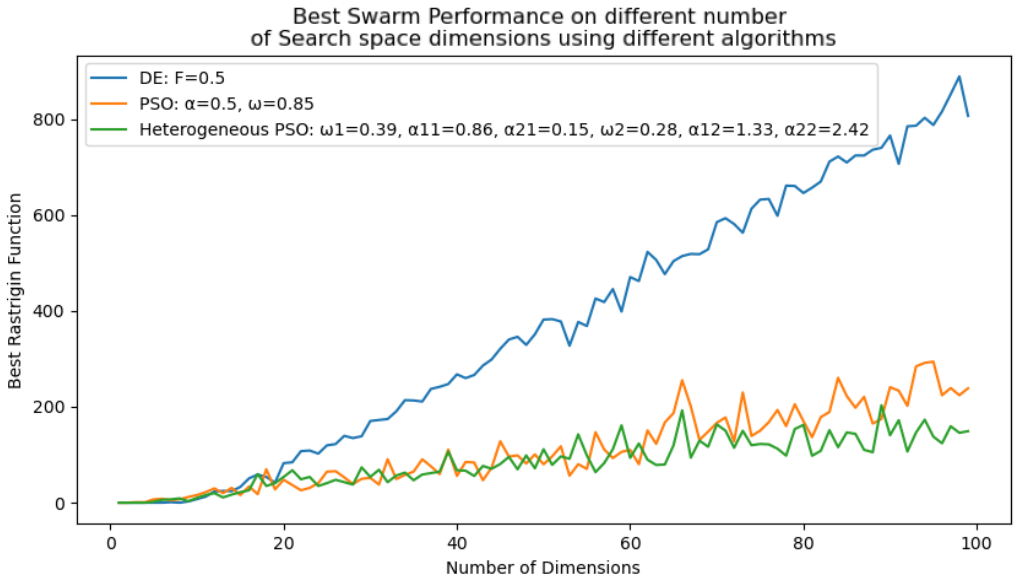

Figure 8: Graph showing the performance of the Rastrigin function using different algorithms (DE, PSO, heterogeneous PSO) across different number of dimensions

The **scalability of DE seems to be worse than PSO**. The graph above shows that even for an optimised *F*, the Rastrigin function value raises above 800 on high dimensions. When PSO uses optimised parameters, this value doesn't raise above 400. Similarly, when we use a disperse parameter heterogeneous PSO the scalability outperforms DE.

DE's failure to compete on scalability with PSO could be caused by the fact that individuals in DE are not allowed to decrease in fitness, making it difficult to escape local minima. This does not occur on PSO, as particles change positions depending on several forces allowing them to go back to positions of lower fitness.

On dimensions above 20, variations of PSO evidently outperforms DE. However, from Figure 8 it is not clear if the performance of DE in lower dimensions is worse than PSO. To inspect this area of performance I compared PSO and DE with the same parameters as before, but only from dimensions 1 to 20.

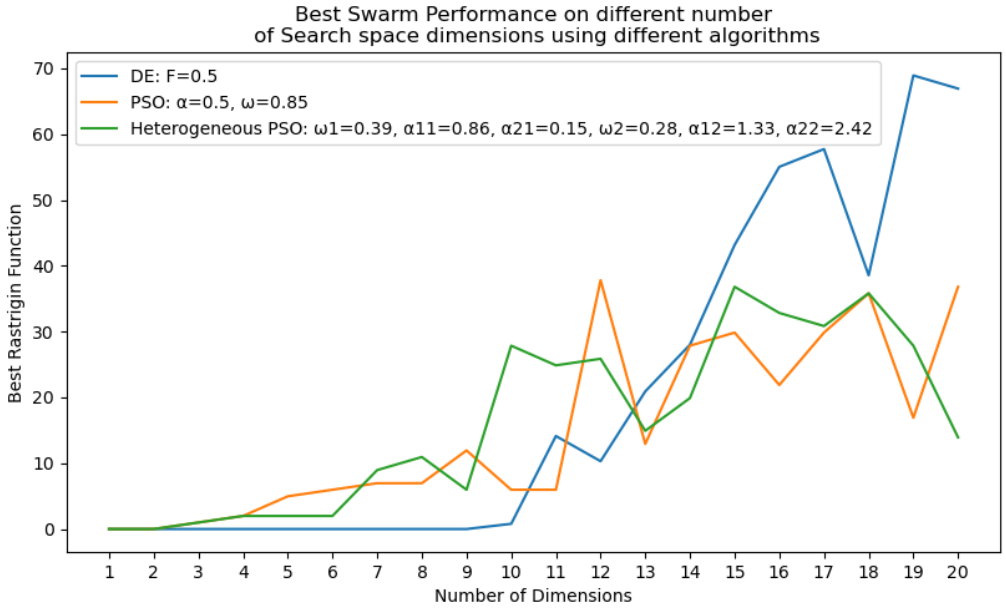

Figure 9: Graph showing the performance of the Rastrigin function using different algorithms (DE, PSO, heterogeneous PSO) on a small number of dimensions

From this graph we can see that DE consistently outperforms PSO in lower dimensions. Up to 10 dimensions, the performance of DE has reached the global optimum, whereas both variations of PSO leave the global optimum after the third dimension. After *d* reaches ten dimensions, as we increase *d,* the performance of DE starts to increase away from the global optimum faster than the different variations of PSO.

In conclusion, **DE seems to outperform variations of PSO in lower dimensions**. However, DE cannot compete with the scalability of PSO. PSO's Rastrigin function results increase slower than they do for DE as dimensions increase.

Problem 5 – Genetic Programming

For this task I used the code from Sipper, M. [1]. Minimal changes were made to apply the code for this problem. The main change was making the objective function the Rastrigin function. For the sake of simplicity, the search space dimension is 1, and the numerical and operation values were restricted to the ones that appear in the Rastrigin function.

For comparison purposes the tree below shows how the Rastrigin function is represented:

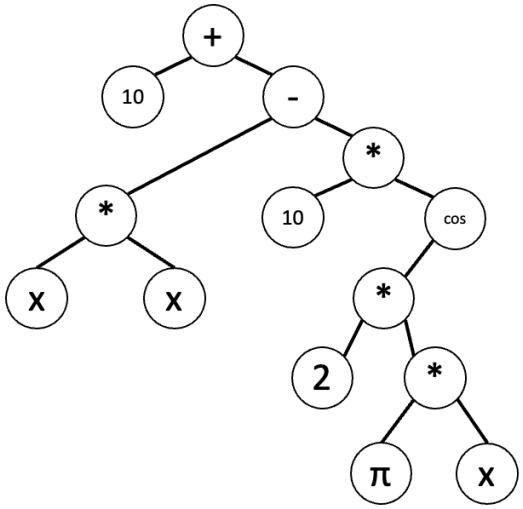

Figure 10: Tree representing the Rastrigin function

After building the dataset with 100 points from the range of -5.12 to 5.12, the GP produced the following tree to predict for the Rastrigin function:

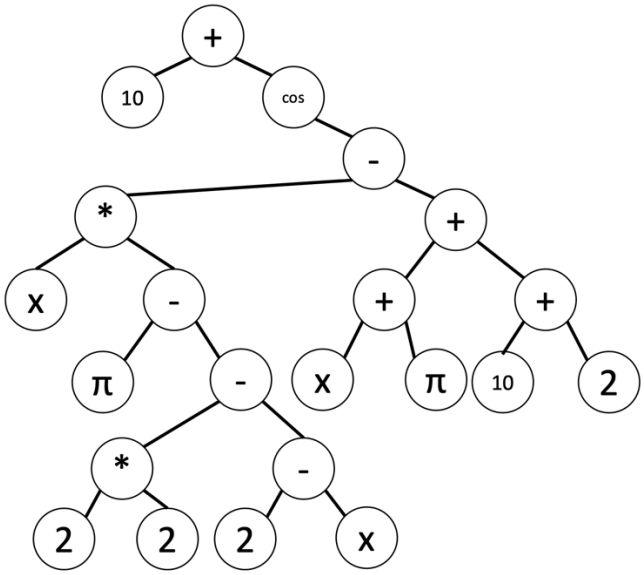

Figure 11: Tree individual with highest fitness in GP algorithm attempting to represent the Rastrigin function

In an equation form it would look like this:

$$\hat{y} = 10 + \cos((x * (\pi - ((2 * 2) - (2 - x)))) - ((x + \pi) + (10 + 2)))$$

The fitness of trees is calculated with the inverse mean absolute error over the normalized dataset *f*. This was the best performing tree with *f* = 0.083. If we compare how this tree produces output for x, to how the Rastrigin function does, the following graph is generated:

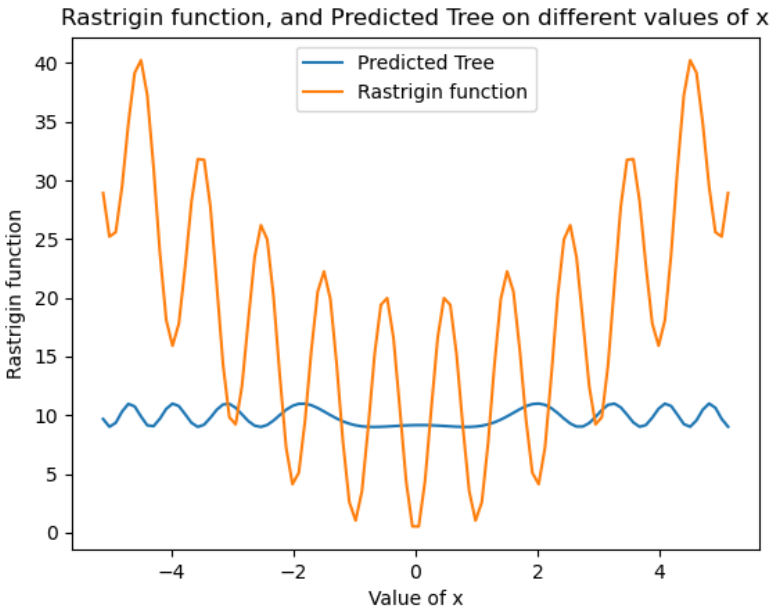

Figure 12: Graph showing the values of Rastrigin function on different values of x, and the values of the predicted tree generated by the GP as a prediction of the Rastrigin function

There are several points of intersection between the two lines, lowering the mean error between the predicted tree and the Rastrigin function, but the overall shape is not matched. It is clear that **the GP with the stated parameters and assumptions wasn't able to perform well** in predicting the shape of the Rastrigin function.

[1] Sipper, M. (2019) *Tiny Genetic programming in python*, *GitHub*. Available at: https://github.com/moshesipper/tiny_gp (Accessed: November 8, 2022).