Natural Computing Assignment

s1805741

October 2022

Contents

Introduction

The following experiments were carried out within a single Jupyter Notebook (`.ipynb`) file. The algorithms themselves are modified versions of the notebooks that were supplied to students in the course materials page of the Natural Computing learn page, with the only exception being in Problem 4, where the Differential Evolution algorithm used is taken from a Scipy library[1].

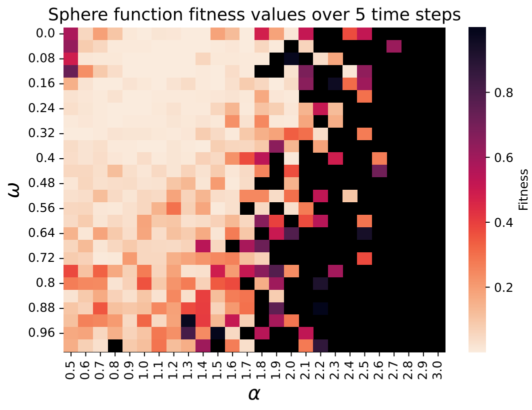
Problem 1: Analysis of Particle Swarm Optimisation

During this analysis, the minimisation of two separate fitness functions were analysed and any differences between their optimal parameter values discussed. The two fitness functions in question were the *Rastrigin*[3] and *Sphere*[2] functions respectively. Each of the functions in the following experiment were tested using a 3-dimensional input vector.

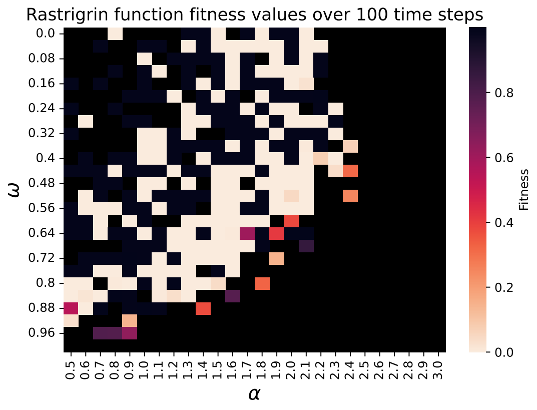
To determine hyperparameter values with which the standard Particle Swarm Optimisation (PSO) algorithm can obtain good results in a short time, a range of values for both the *inertia*, ω, and the *cognitive* and *social coefficients*, α_1 and α_2 respectively, were explored in the ranges of $[0, 1]$ and $[1.5, 2.5]$ respectively. To simplify the problem, the *cognitive* and *social coefficients* were combined into one variable, α. The ranges for either variable were chosen as they (roughly) adhered to the values used in a hyperparameter optimisation study by Kennedy and Eberhart (1995)[4] regarding PSO. The *search range* for the PSO algorithm was consistent throughout the experiments, with a range of $[-5.12, 5.12]$. Similarly, the *population size* was kept constant at 30.

The chosen values need to both produce a sufficiently good fitness level for this minimisation problem (≈ 0) and to do so in a short time. As such, the number of *time steps* taken by the PSO algorithm began at a small value (≈ 5) and increased until a significant number of candidate solutions were found. Although no precise measure was used to determine what defined a 'significant' number of solutions, the general rule of thumb was that the time step limit was increased until a characteristic curve could be identified within the heatmaps generated.

Each run was terminated if the values lead to divergence (in which case the fitness score was marked 1000) or the time step limit was reached, with the best fitness for each run recorded as a heatmap in Figure 1:



(a) Fitness distribution of PSO using the Sphere function, showing emergent candidate solutions after 5 time steps.



(b) Fitness distribution of PSO using the Rastrigrin function, showing emergent candidate solutions after 100 time steps.

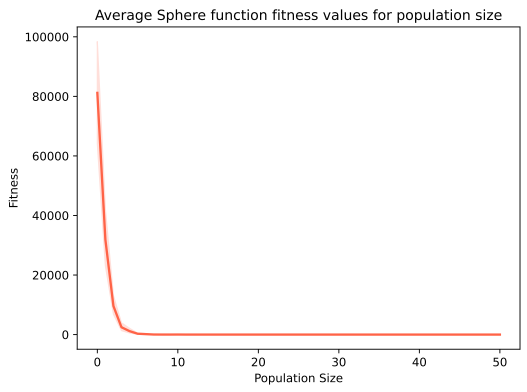
Figure 1

In Figure 1b the Rastrigrin function takes far longer to start producing results when compared to the apparent immediate success of the Sphere function (Figure 1a). Regarding candidate hyperparameter values, the Sphere function has far more success, producing results after only 5 time steps. These solutions tend to exist among the lower values of ω and α, with $\omega = 2.4$ and $\alpha = 1.0$ being an example of a solution scoring a fitness of 0. For the Rastrigrin function, the candidate solutions tend to involve fairly middling values of ω and α, with some solutions appearing with high values of ω (0.8) in combination with low values of α (0.5). There were also some solutions distributed among the low values of ω (0.08) and the middling values of α (2.0). By and large, the Rastrigrin function either diverged or scored poorly in fitness, or obtained a fitness score of 0. An example of parameter settings that scored perfect fitness for the Rastrigrin function is $\omega = 0.48$ and $\alpha = 1.6$. At the respective time step limits for both functions, a characteristic curve begins to form. More evident in the Rastrigrin function, this curve delineates the boundary between good solutions and divergent/poor solutions.

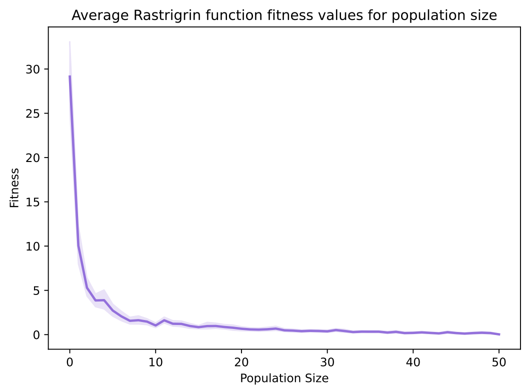
Problem 2: Scaling

For the following experiment, one additional parameter was chosen and its corresponding effect on both the previous fitness functions discussed. The parameter that was explored in this problem was the population size. It must be noted that all the other parameters were kept constant, as they were in the previous experiment, minus the ω and α values, where instead a well-performing combination for each function (as identified in Problem 1) was used. Those values were $\omega = 2.4$, $\alpha = 1.0$ for the Sphere function and $\omega = 0.48$, $\alpha = 1.6$ for the Rastrigrin function.

Similarly to the previous experiment, the search in an individual run was terminated if divergence was encountered. This was unlikely to occur as selected inertia and cognitive & social coefficients were used. However, if divergence had happened, the fitness would have been recorded as -1 and filtered out. A value of -1 was chosen for the penalty as it would separate it from the other potentially high fitness values. Otherwise, the search was terminated once the time step limit was reached. For this experiment, the number of time steps for the Sphere function was 5, and for the Rastrigrin function it was 100. This was to ensure that both functions returned good solutions in a short time, relative to their own historical performances. Each fitness function was tested against the population range multiple times and an average best fitness recorded. 51 repetitions were used as it gave a square matrix output that was easy to manipulate, and also gave enough time for each to produce a fair average result:



(a) PSO fitness distribution using the Sphere function against population size, showing a clear improvement in average fitness over 51 repetitions, with 5 time steps and increasing population size. $\omega = 2.4$; $\alpha = 1.0$; Dimension = 3; Search space = [-5.12, 5.12]



(b) PSO fitness distribution using the Rastrigrin function against population size, showing a clear improvement in average fitness over 51 repetitions, with 100 time steps and increasing population size. $\omega = 0.48$; $\alpha = 1.6$; Dimension = 3; Search space = [-5.12, 5.12]

Figure 2

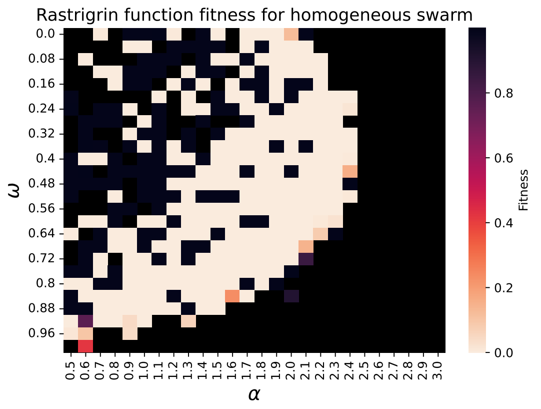
In Figure 2a, we can see that the Sphere function followed a steep downward trend in fitness, with a population approaching 0 causing high fitness values (note that these values are not a result of divergence detection assigning high values to the fitness, but in fact poor performance by the algorithm). From a population size of around 5, the Sphere function showed an almost perfect fitness score. Although it may seem that the Sphere function showed little fitness variation throughout the latter population sizes, the slightest deviation on the graph could mean a large difference in score due to the scale of the fitness axis. Although the graph is good for seeing the general trend, further numerical analysis was needed to show that the Sphere function displayed a huge amount of variation at the smaller population values, though from a population of around 10, showed little variation whatsoever.

In Figure 2b, we can see the Rastrigrin function also follows a similar downwards trend, although not quite as severe as the Sphere function (Figure 2a). The Rastrigrin function benefitted from higher population sizes, showing continuous improvement as the population size grew to 50. This shows that the increase in number of particles for a given swarm aided the algorithm in escaping the potential (and abundant) local minima present in a traversal of the Rastrigrin function.

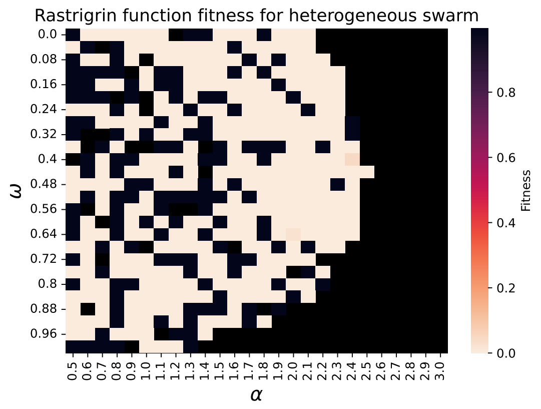
Problem 3: Heterogeneous Particle Swarms

To determine the effect of introducing a secondary particle type, one particle type was assigned the values discovered in Problem 1, and subsequently used in Problem 2 (for the Rastrigrin function, this was $\omega = 0.48$ and $\alpha = 1.6$) and the other particle was assigned the same search space as defined in Problem 1 ($\omega \in$ [0, 1] and $\alpha \in$ [1.5, 2.5]). Note that the cognitive and social coefficients were assigned the same value as each other, as before.

With the introduction of a new particle, the system may once again have the potential to diverge. If this was the case then the PSO algorithm would detect whether any particle is too far from the center, stop the run and return a fitness of 1000 which would then be later filtered from the results. In the usual case where the function converges, the run would terminate once the time-step limit is reached, with the best fitness then being returned and recorded.



(a) PSO fitness distribution with the Rastrigrin function over 300 time-steps with a homogeneous swarm. All particles involved alter per the axis values shown. Population size = 30; search space = [-5.12, 5.12]; dimension = 3.



(b) PSO fitness distribution with the Rastrigrin function over 300 time-steps with a heterogeneous swarm. Half of the particles in the population alter per the axis values shown, the other half have constant values of $\omega = 0.48$ and $\alpha = 1.6$.
Population size = 30; search space = [-5.12, 5.12]; dimension = 3.

Figure 3

For the most part, both the heterogeneous and the homogeneous swarms gave similar results, both showing a lack of willingness to find solutions that exist in between perfect and poor fitness scores - with the homogeneous swarm being slightly more open to the idea. The decision to keep one particle constant and to keep the coefficients the same relative to their particle type was made to reduce the number of variables in the experiment and make the results far more readable. However, in choosing values of ω and α for the constant particle to be the successful, low fitness producing values meant that the heterogeneous swarm had a slight advantage over the homogeneous swarm in terms of finding successful solutions.

The homogeneous swarm (Figure 3a) is an obvious continuation of the fitness distribution found in Figure 1b, with this run's curve being more defined due to the greater time step limit. The fitness values recorded have a seemingly binary nature, and exist mainly among $\omega \in [0.4, 0.96]$ and $\alpha \in [0.5, 2.4]$. As with the results for the lower time steps (Figure 1b), high values of α ($\alpha \geq 2.5$) return no, or few, results.

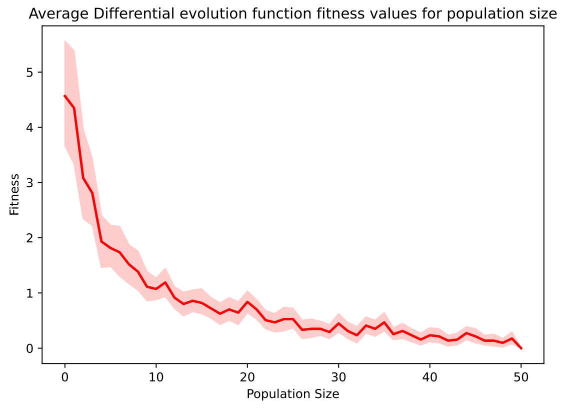
Evaluating the performance on the heterogeneous swarm, we can see that it followed the same curve as the heterogeneous swarm, with solutions tending to only exist within the convex structure (Figure 3b). However, with the heterogeneous swarm the results seem far more binary than the already very binary homogeneous counterpart, with either very good, or very poor results. As expected, the heterogeneous swarm found far more 'excellent' solutions, with a large portion of the lower ω and α values finding solutions, where the homogeneous swarm did not. Similarly to the homogeneous swarm, large values of α ($\alpha \geq 2.5$) harboured no good fitness values.

Problem 4: Differential Evolution

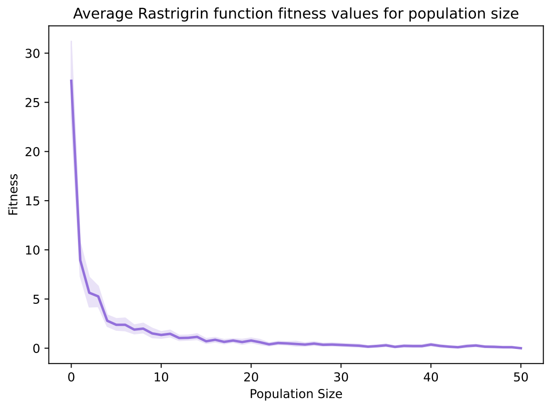
For this problem, the Differential Evolution (DE) algorithm, first implemented by Storn and Price (1997)[5] was compared against a PSO algorithm competing in the same problem. An implementation of the DE algorithm is available on Scipy[1] for use in Python programs, and was used here.

The DE fitness function takes two main arguments, the *mutation rate* and the *recombination rate*. These were set to 0.8 and 0.9 respectively, as they closely resembled the suggested values in the documentation for the Scipy DE algorithm [1]. To compare the performance of the Rastrigrin fitness function in the PSO and DE implementations, both were run with the same parameter settings (excluding the algorithm specific parameters, such as mutation rate or social coefficient) and repeated 51 times to record a truthful average fitness score for both algorithms. Both algorithms were tested against varying population sizes, from 1 to 50. A time-step limit of 300 was chosen as it allowed enough time for either algorithm to produce results characteristic of their individual approaches without prematurely ending their respective searches.

As before, the search would terminate if divergence was detected. This was handled externally by the Scipy implementation, but for the PSO function, it was the same as before - the function would detect divergence and filter out the result. Otherwise, the best fitness obtained within the time step limit was recorded and averaged over the runs. It can be noted that neither experiment encountered divergence for this particular problem.



(a) DE with Rastrigrin function average fitness distribution over 300 time-steps with varying population size and 51 repetitions. Mutation rate = 0.8; Recombination rate = 0.9; Search space = [-5.12, 5.12]; Dimension = 3.



(b) PSO with Rastrigrin function average fitness distribution over 300 time-steps with varying population size and 51 repetitions.
ω = 0.48; α = 1.6; Search space = [-5.12, 5.12]; Dimension = 3.

Figure 4

There appears to be far more variance in the DE algorithm's performance (Figure 4a) than the PSO algorithm (Figure 4b) but this is largely down to the scale of the graphs; numerically their variances are largely the same. However, it is clear to see that the DE algorithm far outperforms the PSO algorithm at the smaller population sizes, especially at a population size of 1. This is likely down to PSO being inherently swarm based, where a single particle will not benefit from the advantages that swarm optimisation entails. After this, there does not seem to be much difference between the performance of the two algorithms, both tend towards an average fitness of 0 when reaching the upper population limit, in this case 50 for both algorithms.

Problem 5: Genetic Programming

To begin generating the Rastrigrin function through Genetic Programming (GP), the function must be first broken down into its terminals and non-terminals. For the sake of ease, the dimensionality of the input vector was reduced to 1, this meant that there was no summation to worry about in the set of functions to define the non-terminals - thus simplifying the equation to:

$$10d + \sum_i (x_i^2 - 10\,cos(2\pi x_i))$$

$$\xrightarrow{d=1} 10 + (x^2 - 10\,cos(2\pi x))$$

While the set of legal numerical values could be reduced to the set of relevant numbers ($[x,\ 10,\ 2,\ d,\ \pi]$), it was noticed that this set could be reduced further. Firstly, the dimensionality (d) is expanded into the fitness function, so can be removed from the set, furthermore, 2 and π only exist in the equation together, so can be set as a single value, 2π. The resulting set of terminals for the GP algorithm is then $[x,\ 10,\ 2\pi]$.

The termination criteria for this experiment is defined by the GP algorithm ending when a solution is found with a perfect fitness score (fitness = 1.0), or the maximum number of generations is reached.

The parameter settings used for this particular experiment were a mutation rate of 0.1, a crossover rate of 0.9, a population size of 600, a minimum and maximum starting tree depth of 2 and 5 respectively and a generation cap of 250. This particular generation cap was used as a buffer window for ensuring the function was generated. In practice the function was generated well within these bounds for the 1-D case - on average after around 40 generations. There was also a separate parameter regarding the selection process of the GP algorithm - the tournament size. This was set to $\frac{1}{30}$ of the total population size, which worked well in the 1-D case. The tournament selection process was chosen in favour of the roulette selection process as it was far easier to implement in Python, as one was supplied for us.

The GP algorithm, although allowing for fine tuning of all the other parameters, was not built with unary functions in mind - every non-terminal is expected to take two arguments, even when two do not exist (for example in a *cosine* function). To circumvent this, the *cosine* function was defined as taking two variables and returning the cosine of the product of these variables:

$$cos(x, y) \rightarrow cos(x * y)$$

This design choice was taken as it simplified the implementation of the GP algorithm, and as the cosine function used in the Rastrigrin function was only ever operating on products of numbers, was deemed suitable.

The algorithm was run with these settings 10 times and, on average, the GP algorithm managed to recreate the algorithm, or come incredibly close to (≤ 0.05 fitness score from a perfect score of 1), within the 250 generations - sometimes obtaining the solution as early as 8 generations (Appendix Figure 5). This shows that the parameter settings used for this task are suitable for recreating the 1-D Rastrigrin function using a GP approach. In order to ensure the parameters were satisfactory and the results weren't obtained by chance, further repetitions could be carried out; this was not done given time constraints as each run took a number of minutes. To further explore the GP algorithm's performance in recreating the Rastrigrin function, the same process was used for the 2-D case:

$$10d + \textstyle\sum_i (x_i^2 - 10\,cos(2\pi x_i))$$

$$\xrightarrow{d=2}\ 20 + x_1^2 - 10\,cos(2\pi x_1) + x_2^2 - 10\,cos(2\pi x_2)$$

The algorithm struggled with this new setup, using 250 generations to find a best fitness, which rarely approached a perfect fitness and almost always stagnated around a smaller value. However, the algorithm used was not designed to take inputs exceeding 1 dimension. The extra dimensional values were calculated inside the fitness function itself. Although this is not how it would normally function, it should not have affected the performance in any way as the values are still generated within the same range and distributed correctly among the variables. It also took a far longer time to compute each solution; while the 1-D function was computed within 3 minutes, this 2-D case took upwards of 10 minutes to calculate half of the generations. Therefore, it appears to be the case that in order for a good solution to be found within a reasonable time the GP algorithm requires fine tuning depending on problem assigned.

References

[1] Scipy: Differential evolution.

[2] Virtual library of simulation experiments: Sphere function.

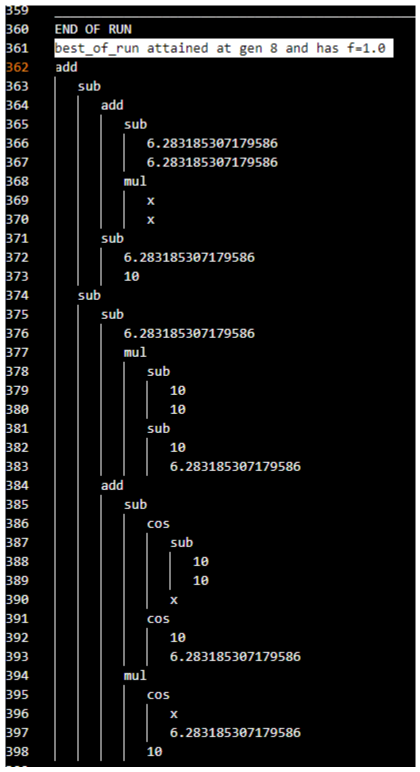
[3] Leonard Andreevič Rastrigin. Systems of extremal control. *Nauka*, 1974.

[4] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.

[5] Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

Appendix

Extra: Genetic Programming



```
END OF RUN
best_of_run attained at gen 8 and has f=1.0
add
│  sub
│  │  add
│  │  │  sub
│  │  │  │  6.283185307179586
│  │  │  │  6.283185307179586
│  │  │  mul
│  │  │  │  x
│  │  │  │  x
│  │  sub
│  │  │  6.283185307179586
│  │  │  10
│  sub
│  │  sub
│  │  │  6.283185307179586
│  │  │  mul
│  │  │  │  sub
│  │  │  │  │  10
│  │  │  │  │  10
│  │  │  │  sub
│  │  │  │  │  10
│  │  │  │  │  6.283185307179586
│  │  add
│  │  │  sub
│  │  │  │  cos
│  │  │  │  │  sub
│  │  │  │  │  │  10
│  │  │  │  │  │  10
│  │  │  │  │  x
│  │  │  │  cos
│  │  │  │  │  10
│  │  │  │  │  6.283185307179586
│  │  │  mul
│  │  │  │  cos
│  │  │  │  │  x
│  │  │  │  │  6.283185307179586
│  │  │  │  10
```

Figure 5: The proof-of-result for the Genetic Programming algorithm recreating an equivalent Rastrigin function after 8 generations. Note that 6.283... here corresponds with 2π.