Coursework 2 - Implementing and Comparing Genetic Algorithm and Particle Swarm Optimisation

F21BC: Biologically Inspired Computation

Barthelemy Traxel, Peter Cseh

H00412986, H00326306

Table of Contents

[1. Introduction 2](#_Toc120590071)

[2. Implementation 2](#_Toc120590072)

[2.1 Genetic Algorithm 2](#_Toc120590073)

[2.1 Particle Swarm Optimisation 3](#_Toc120590074)

[3. Results 3](#_Toc120590075)

[4. Discussion and Conclusions 4](#_Toc120590076)

[5. References 4](#_Toc120590077)

# Introduction

The task was to implement Genetic Algorithm (GA) and Particle Swarm Optimisation (PSO) optimization algorithms from scratch and to carry out some experiments to see how they compare when evaluated on the CEC 2005 test functions (Suganthan et al., 2005). These test functions are used to assess optimization functions on an artificial landscape. The goal of the optimization algorithms was the find the global minimum (optimum value) in these test functions.

# Implementation

The implementations were based on the pseudocode from Luke (2016). The logic for the algorithms can be found in GA.py and PSO.py.

The following test functions were picked to test the algorithms with the following properties in mind:

* F2: Shifted Schwefel’s Problem 1.2
* F4: Shifted Schwefel’s Problem 1.2 with Noise in Fitness
* F8: Shifted Rotated Ackley’s Function with Global Optimum on Bounds
* F13: Shifted Expanded Griewank’s plus Rosenbrock’s Function (F8F2)
* F17: Rotated Version of Hybrid Composition Function with Noise in Fitness
* F24: Rotated Hybrid Composition Function

The aim was to cover a lot of different types of properties. F2 and F4 are both unimodal, without and with noise. All the rest are multi-modal. F8 is a single function, F13 is an expanded function and F17 and F24 are composite. F17 has a Gaussian noise in its fitness and for this reason, this is in-deterministic. All the other functions are deterministic. F17 and F24 are composite and due to the nature of the function, it has a lot of spikes in their artificial landscape which results in a high number of local optima. F8 has its local optima on its bounds.

## Genetic Algorithm

The genetic algorithm was implemented with elitism (Luke, 2016, pp. 46-47.). The code follows the same structure as the pseudocode, however, it had to be tweaked to allow multiple elites. The pseudocode’s line 10-11 is responsible for saving the best results after assessing the fitness. This was implemented differentially by using Python’s inbuilt sorting functions.

The algorithm has the option to use decreasing mutation rate. The program slowly decreases the mutation rate per generation if this is used. I.e. if the mutation rate is 80% then for the first generation the rate is unchanged, at half-point it's halved for 40% and the last generation is near 0%. The advantage of using decreasing mutation rate is to have a high mutation rate at the beginning which helps the algorithm to explore the artificial landscape and also avoid local optimums but narrows down the randomness as it progresses forward which helps to focus on attired solutions towards the last generations without scrambling it with high mutation.

Tournament selection is used with adjustable tournament size, 2 on default as per Luke (2016, pp. 45.).

Crossover percentage can be defined as well, this parameter gives the probability of crossover happening between parents. By default, it is 95%.

## Particle Swarm Optimisation

The algorithm for Particle Swarm Optimisation was implemented with the use of informants (Luke, 2016, pp. 55-57.). The code follows the same structure as the pseudocode, however, it had to be tweaked to allow the implementation of the informants. The code has been realised using object orientated programming.

The algorithm is ensured to be able to run on every problem bounded or not of the CEC 2005, all hyperparameters are configurable. The number of iterations is controlled by a maximum number of iteration and a precision which when reached stops the program, avoiding useless computation. An option that could be added in the future could be to add a bouncing option when particle reach the boundaries of a problem.

# Results

The execution and analysis of the optimization functions can be found in the GA\_notebook.ipynb and PSO\_notebook.ipynb notebooks. We strongly recommend reading through the notebooks as those contain all our results analysed and explained.

The Genetic Algorithm was run in two iterations. First, we explored how GA evaluates a set of configurations using only lower numbers of generations. The optimizer performed the best on F8 and F13.

Each test function achieved the best score using 150 generations which was the maximum in the first iteration, so we decided to increase the number of generations to see how they improve. For the 2nd iteration, we kept the configuration of the best settings per test function. Each test function has two best scores. Best fitness and best average fitness.

Overall, the algorithm improved by increasing the number of generations and size of the population. The greatest improvement happened on the first two uni-modal functions (F2, F4). For the 2nd iteration, there was a notable improvement in the extended multimodal function (F13). However, on this function, the optimizer did well already in the 1st iteration and the change wasn’t as enormous as it was for the two uni-modal functions. The single function (F8) didn’t improve for the 2nd iteration, the best fitness on this test function did decrease slightly. The algorithm achieved a similar result as it did with the lower generation number in the 1st iteration, which implies that we found a good solution with a low error rate and low generations. The composite functions (F17, F24) did improve too, but their error was too high, with these configurations GA couldn’t a suitable solution for the test function. The high number of local optima that these composite functions have suggests that the optimizer algorithm got stuck in a local optimum. This can be avoided by increasing the population number.

Through the experiments done in the PSO notebook, we saw that augmenting the number of iterations for PSO improves the average results, however, using fewer iterations allows us to obtain better results as the particle are less trapped into local optima. The swarm size can be optimal at around 35 for certain problems and around 100 for others. Too big swarms tend to augment the computation time without improving the results. On F2 and F4 the PSO algorithm improved greatly by increasing the generation and population numbers. The PSO optimizer gave the best results on F13 and F8. F17 and F24 require much more computation than F13, F8 or F2 but can reach better average results. The cause of this could be because the tuning of the best parameters was done using F17, for more on this please see the Hyperparameters exploration paragraph in PSO\_notebook.ipynb. Previous experiments showed that with other parameters PSO can find the global optima for F2, F4 and F8, for more details on this please see Additional test in PSO\_notebook.ipynb.

# Discussion and Conclusions

Overall, the two optimiser algorithms behaved similarly on the test functions. By increasing the number of generations, the best fitness for the test functions improved in the same way. F2 and F4, the uni-modal algorithms improved drastically by increasing the number of runs, however, GA achieved lower error for these uni-modal functions as the PSO optimizers. From this, we conclude that GA performed better on uni-modal functions.

For both algorithms, the best performances were achieved on F8 and F13. PSO performed slightly better on F13 because it had a lower error, however, in terms of speed it took the same time. For PSO it took 1.2 seconds over 250 iterations and for GA 1.2 over 1000 generations. It is important to mention the difference between the two algorithms that doing 250 iterations on PSO took approximately the same amount of time as 1000 generations in GA with these best configurations. The computation times of these algorithms are subject to a lot of different hyperparameters. It’s important to note that the number of generations is not comparable to the number of iterations on its own when we speak of performance. For F8 the algorithms performed similarly, it didn’t improve much by increasing the number of generations.

For the composite functions F24 and F17 both algorithms performed poorly. However, for F17 the PSO achieved lower error, but nothing near the optimal.

As a conclusion for comparing the algorithm in hindsight, we think the best way is to time the algorithms and run them for the same amount of time, with similar hyperparameters and compare the results. Our approach was to limit the algorithm by running it for a given number of iterations, but it would have been wiser to use a time limit instead. By using a time limit it's easy to compare the algorithms, as the one with better fitness is the one which performed better.

# References

Luke, S. (2016). *Essentials of Metaheuristics.* 2.3 ed. [online] Lulu, pp.31–58. Available at: https://cs.gmu.edu/~sean/book/metaheuristics/ [Accessed 27 Nov. 2022].

Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y., Auger, A. and Tiwari, S. (2005). Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on RealParameter Optimization. *Natural Computing*, 341-357.