Biologically Inspired Computation
Coursework 2

Alexandre Kha and Raphaël Valeri

November 29, 2022

Introduction

In this report, we describe the implementation and the performance of 2 bio-inspired optimization algorithms, the Genetic Algorithm (GA) and the Particle Swarm Optimization (PSO), designed almost from scratch, in Python language.

For a given problem, both of these algorithms search for a solution that has an optimal (maximum or minimum) fitness function, among a population of solutions (also called individuals). However, in the GA, the population of solutions evolves according to the Darwinian Theory principles, like natural selection and genome mutation; while in the PSO, the population of solutions can be seen as a swarm of individuals, which move according to cognitive and social factors. To compare these algorithms, we search for their optimal hyperparameters on different benchmark functions to minimize, and provide a discussion on how suited these methods are to certain problems. A user will have the possibility to try these algorithms and follow the evolution of their performances in a terminal, where options of parameters will be provided to them.

1 Program development rationale

Object Oriented Programming (OOP) has been chosen to unify related entities and methods under objects. In a lesser measure, OOP also easily enables to apply the Open/Closed principle, as code extension can be done through inheritance. Thus, for example in the case of the GA, a variety of genetic operations are available through the attributes and methods of the Selection, Crossover and Mutation classes. In both our GA and PSO implementations, individuals are coded as real vectors. In the case of this study, each of their coordinates corresponds to a dimension of the benchmark functions.

2 Methods

2.1 Selection of benchmark functions

In order to explore the performances of our optimization algorithms, three benchmark functions have been selected to be optimized with the GA and the PSO algorithms from [1]. Their name, expression and characteristics are gathered in the Table 1. They have been chosen for their complexity and the diversity of their landscapes. The Rastrigin function is a highly multi-modal function which makes the convergence towards the global optimum while avoiding local optima very challenging. It is separable, i.e. it can be expressed as a sum of independent uni-variable functions. This property can eventually lead to a variable-by-variable minimization. The Rosenbrock function is also a multi-modal function (with a dimension higher than 3), but the global minimum is located at the bottom of a parabolic flat valley, with a local optimum at one side [2]. It is not separable, which means variables cannot be optimized independently. Ultimately, we consider the Griewank function, which is, like the Rastrigin function, extremely multimodal, but with relatively less widespread minima and a smaller decreasing rate around the center. In contrast to Rastrigin, it is not separable.

Name	Expression	Separable	Dimension	Boundary	Global optima
Rosenbrock	$f_1(x) = \sum_{i=1}^{D-1}(100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2)$	No	10	$[-100, 100]^D$	$f_1(1, 1, ..., 1, 1) = 0$
Rastrigin	$f_2(x) = 10D + \sum_{i=1}^{D}(x_i^2 - 10cos(2\pi x_i))$	Yes	10	$[-5.15, 5.12]^D$	$f_2(0, 0, ..., 0, 0) = 0$
Griewank	$f_3(x) = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} cos(\frac{x_i}{\sqrt{i}}) + 1$	No	10	$[-600, 600]^D$	$f_3(0, 0, ..., 0, 0) = 0$

Table 1: Selected benchmark functions

2.2 PSO implementation and hyper-parameters tuning

2.2.1 Implementation of the PSO algorithm

The PSO algorithm uses a population of particles which each has a position in the search space and a velocity. The key point of this algorithm is the update of the velocity of each particle. Different approaches have been used in the literature since the emergence of this well-known optimization technique in 1995 [3]. The implemented algorithm is the one from [4], which takes into account a cognitive and a social component by considering for each particle a number of informants. The best position seen by any of these informants (x^+) is used to tweak the velocity of the particle into this direction. The velocity is also tweaked in the direction of the best position seen by the particle (x^*), and the best position seen by all the particles of the swarm x'. Finally, the equation of the updated velocity for the dimension i at time $t+1$ given time t is the equation 1, with $u_k(t) \sim \mathcal{U}([0,1])$.

$$v_i(t+1) = \alpha v_i(t) + \beta u_1(t)(x_i^*(t) - x_i(t)) + \gamma u_2(t)(x_i^+(t) - x_i(t)) + \delta u_3(t)(x_i'(t) - x_i(t)) \quad (1)$$

The implementation offers two possibilities for the number of informants of each particle. The user can set a number of informants which will be used for each particle, or the number can be randomly chose at each iteration, then, for each case, the informants are randomly selected from the swarm.

This algorithms have several hyperparameters which impact its performances and need to be optimized for each particular problem. There are the four acceleration coefficients: α (inertia weight), β (cognitive factor), γ (social factor) and δ, the step size ϵ, and the swarm size. A lot of researchs have been done and published to optimize the values of these hyperparameters. First, it appears that an adaptive inertia weight is beneficial for the convergence of the algorithm and several techniques can be used and are gathered in [5]. The same reasoning of adaptive coefficients has been used with the cognitive and social coefficients [6]. The optimizations used in our work are linearly decreasing or increasing coefficient and are presented in appendix A (table 6).

2.2.2 Hyperparameters investigation

As the number of possible hyperparameters values is quite huge, the method used in this work is to test the improvement of the results using the implemented adaptive coefficients and the values advised in the literature reported in the table 6 (Experiment 1 in the table 2). Then, the other hyperparameters - mainly the weight for the global best position seen by the swarm (δ), the number of informants and the step size - are investigated. Each experiment presented in the table 2 aims to find the optimal value for one hyperparameter among a range of possible values. The result of this investigation for the Griewank function is presented in appendix B (figure 1).

Exp	n_{pop}	$n_{informants}$	α	β	γ	δ	ϵ
E.1	40	random	$\{0.7, 0.9, 1, 1.1\}$, TVIW	2, TVAC	2, TVAC	0.4	1
E.2	40	random	TVIW	TVAC	TVAC	$\{0, 0.2, 0.4, 0.6, 0.8, 1\}$	1
E.3	40	random	TVIW	TVAC	TVAC	0.4	$\{0.6, 0.8, 1, 1.2\}$
E.4	40	$\{5, 10, 15, 20, 25, 30\}, random$	TVIW	TVAC	TVAC	0.4	1

Table 2: Experiments for PSO hyperparameters investigation

2.3 GA implementation

The GA creates new generations of solutions through methods of parents selection, crossover and mutation. Popular selection techniques have been implemented, among which the **Tournament** selection where winners of a fitness tournament are more likely to be selected (probability fixed to 0.9), and the **Wheel-Roulette (WR)** selection, where a chromosome's selection chance is proportional to its fitness.

Crossovers, occurring with probability *cp*, mix parent genes to produce a child. Here, some typically used on bit-encoded chromosomes, were implemented, like the Uniform and k-points crossovers [7]. BLX-α, specific to

real-encoded chromosomes, has also been implemented [8]. It computes a child's gene following $\mathcal{U}\big(\big[x-\alpha.|y-x|,\ y+\alpha.|y-x|\big]\big)$, with $x<y$ the 2 parents genes.

Mutation is done through Gaussian shrinkage. At generation t, a mutation occurs with probability mp on each gene, according to :

$$child_t[gene] \leftarrow child_t[gene] + \mathcal{N}(0,\ V_t) \tag{2}$$

V_t is either, decreasing linearly from a specified V_0, or auto-scaled on the population ($V_t = \underset{gene}{max}\big|pop_t[gene]\big|$). Our work also implements a periodic resetting of the population, when the fitness remains too high. This feature allows to escape some inconvenient local optima or too flat landscapes [9].

2.3.1 Hyperparameters investigation

The maximum number of generations has been fixed to 3000. We take at first (but eventually change if underperformance is constated) $cp = 80\%$ and $mp = 5\%$, typical values suggested by literature [10], which ensure genome mixing, as well as a chance to escape premature convergence. The number of selected parents will be 20% of the population size. Elitism (keeping best individuals) and culling (elimination of the weakest ones) are taken at first at 5 individuals ($elites = cull = 5$) to avoid fitness divergence and still preserve genetic diversity. For the other hyperparameters, we display their range of search in Table 3. Performance will be measured by taking the mean fitness, the standard deviation as well as the best and worst fitness values over 10 trials.

Pop. size	Selection method	Crossover type	Mutation variance (cf (2))
$\{40, 200, 500, 1000\}$	Tournament ($size \in \{3, 5, 10\}$) or WR	Uniform, k-points ($k \in \{1, ..., 4\}$), or BLX-α ($\alpha \in \{0.1, 0.15\}$)	Auto-scaled or $V_0 \in \{5, 10, 15, 20\}$

Table 3: Hyperparameters investigated for the GA

3 Results

We present in the folowing tables the results gathered during the test phase. The hyperparameter values used in these tests for the PSO and GA algorithms are the ones found to be the optimal during the hyperparameter investigation for each function. We consider that a value is equal to 0 when it is inferior to 10^{-6}. For the GA, a population size of 40 gave the most notable trade-off between computation time and optimal fitness.

Benchmark	Pop. size	$n_{informants}$	α	β	γ	δ	ϵ	Mean value	Std	Best fitness	Worst fitness	Min iteration
Rastrigin	40	20	TVIW	TVAC	TVAC	0.6	1	13.7446	11.2996	2.9848	35.8894	1223
Rosenbrock	40	5	TVIW	TVAC	TVAC	0.6	0.8	1.1960	1.8268	0.0	3.9865	1298
Griewank	40	random	TVIW	TVAC	TVAC	0.2	0.8	0.0541	0.0157	0.0271	0.0761	193

Table 4: PSO best results over 10 trials of 3000 iterations

Benchmark	Pop. size	Selection	Crossover	cp	Mutation variance	mp	Mean value	Std	Best fitness	Worst fitness	Min iteration
Rastrigin	40	Tournament ($size = 5$)	1-point	80%	Auto-scaled	5%	0	0	0	0	120
Rosenbrock	40	Tournament ($size = 10$)	2-points	80%	Auto-scaled	10%	15.3781	30.053	0.0075	104.3061	3000
Griewank	40	Tournament ($size = 10$)	1-point	70%	Auto-scaled	5%	0.0116	0.0094	0	0.0225	140

Table 5: GA best results over 10 trials of 3000 generations

4 Discussion

We can see that the performances of the two algorithms are not similar and that the specificities of the fitness functions seem to play a role in this difference. While the GA performs well, robustly - as seen with the low variance of results - and rather rapidly, on the highly multimodal Rastrigin and Griewank, the PSO algorithm struggles to reach the global optimum of Rastrigin. It is probably the most complex function for PSO to optimize because of the huge number of local optima, in which we could infer that a lot of particles fall and attract the others because of the social factors of the swarm. However, the GA has considerable difficulty in minimizing Rosenbrock. Its optimization has high variance, which can be explained by chromosomes prematurely converging to the local optimum or high value flat hills, not providing enough information to escape them [2]. On the contrary, PSO succeeds in one run in finding the global optimum. The dispersion of the particles at the beginning of the algorithm and the attraction towards a better solution is probably beneficial in a flat valley like the one of Rosenbrock. Indeed, it gives information of a better position, that the GA has not.

Globally, the GA seems to perform better than the PSO on this set of benchmark functions, regarding the values of fitness function and the time spent to reach them. Even if the PSO algorithm converges more than two times faster for Rosenbrock, the number of iterations needed for the GA to converge towards a minimum is ten times lower than the PSO for Rastrigin, and quite similar for Griewank. Moreover, the time spent by the GA to run an iteration is less than the one of the PSO, which requires more time to set a list of informants for each particle, get its best informant and update each component of its velocity. This observation joins the one of the work [11], which concludes that the GA performs faster and better than the PSO. Compared to this publication, this work takes into account some possible optimizations of the PSO algorithm, like adding informants and adaptive weights which should reduce the difference of performance between these two techniques (but increase computational time), which is not really the result we can observe here. As already mentioned, here with Rastrigin, the informants can decrease the probability of exploration and so of success, and the sensibility of the weights could also explain this and may require more investigations to be really optimal.

Finally, we should note that, as underlined by [12], performance of the algorithms may depend heavily on the choice and the tuning of the hyperparameters. One reason why the GA outperforms slightly the PSO in this case, might be its superior number of hyperparameters, giving more possibilities to build solutions.

Conclusion

In conclusion, the GA and PSO prove themselves to be efficient optimization techniques on non-trivial problems, as confirmed by the results gathered on our three benchmark functions. The hyperparameters investigation seems to show that crossover rate, mutation rate, and the number of points of the crossover, are the most decisive parameters for the GA, given a problem. Convergence towards a local minimum or a flat surface is still regularly constated on Rosenbrock and Rastrigin though. Concerning the PSO, the balance between cognitive and social components is very sensitive, and the linearly decreasing value for the first and increasing value for the second aim to lead to more individual exploration before being influenced by the informants. Moreover, the weight for the global best particle seems to be also quite important. The performance of the GA and PSO is comparable, even if the former converges slightly faster and with more robustness than the latter here. We should note however that tests on these 3 functions alone do not allow us to provide absolute conclusions over the outperformance of one of the algorithms, or the link between performance and modality or separability of the benchmarks. [13] has tested both the GA and PSO on 30 benchmark functions and demonstrated that PSO was more efficient on most of these problems. But both displayed good results independently of the separability and the modality of the functions. Our work eventually raises the fact that the GA has more difficulty escaping from flat-valley-like surfaces, while the PSO can struggle more with extremely multimodal problems.

Moreover, one result which has not been reflected in our work, is that the dimensionality of a problem can alter the performance of these algorithms. Final tests in 30 dimensions have shown that the GA has now a greater mean value for Griwank, about 0.3, meanwhile PSO succeeds in finding the global optimum with a mean of 0.0091 over 238 iterations. So far, we have reviewed only one study, [14], demonstrating the better and faster performance of the PSO on high dimensional problems, in the specific case of an integer programming problem.

References

[1] Ponnuthurai Suganthan, Nikolaus Hansen, Jing Liang, Kalyan Deb, Ying-ping Chen, Anne Auger, and Santosh Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *Natural Computing*, 341-357, 01 2005.

[2] Jian Ma and Haiming Li. Research on Rosenbrock Function Optimization Problem Based on Improved Differential Evolution Algorithm. *Journal of Computer and Communications*, 07(11):107, November 2019. Number: 11 Publisher: Scientific Research Publishing.

[3] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4, 1995.

[4] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[5] Ahmad Nickabadi, Mohammad Mehdi Ebadzadeh, and Reza Safabakhsh. A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied Soft Computing*, 11(4):3658–3670, June 2011.

[6] M. Senthil Arumugam and M. V. C. Rao. On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems. *Applied Soft Computing*, 8(1):324–336, January 2008.

[7] A. J. Umbarkar and P. D. Sheth. Crossover operators in genetic algorithms:a review. In *Soft Computing Models in Industrial and Environmental Applications*, 2015.

[8] M. Takahashi and H. Kita. A crossover operator using independent component analysis for real-coded genetic algorithms. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 1, pages 643–649 vol. 1, 2001.

[9] Alex S. Fukunaga. Restart scheduling for genetic algorithms. In *Parallel Problem Solving from Nature*, 1998.

[10] M. Srinivas and L.M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, 1994.

[11] Seng Poh Lim and Habibollah Haron. Performance comparison of genetic algorithm, differential evolution and particle swarm optimization towards benchmark functions. In *2013 IEEE Conference on Open Systems (ICOS)*, pages 41–46, 2013.

[12] Karl O Jones. Comparison of genetic algorithms and particle swarm optimization for fermentation feed profile determination. In *Proceedings of the CompSysTech*, pages 15–16, 2006.

[13] Atyabi A. Ab Wahab MN, Nefti-Meziani S. A comprehensive review of swarm optimization algorithms. 2015.

[14] F D Wihartiko, H Wijayanti, and F Virgantari. Performance comparison of genetic algorithms and particle swarm optimization for model integer programming bus timetabling problem. *IOP Conference Series: Materials Science and Engineering*, 332(1):012020, mar 2018.

[15] Y. Shi and R.C. Eberhart. Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950 Vol. 3, July 1999.

[16] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, 2004.

Appendices

A Adaptive weights of PSO

The implemented PSO algorithm offers the possibility to the user to use adaptive coefficients as it is presented in the following table 6.

Strategy	Expression	Advised values
Time varying inertia weight (TVIW) [15]	$\alpha(t) = (\alpha_i - \alpha_f)\frac{t_{max}-t}{t_{max}} + \alpha_f$	$\alpha_i = 0.4,\ \alpha_f = 0.9$
Time varying acceleration coefficients (TVAC) [16]	$\beta(t) = (\beta_i - \beta_f)\frac{t_{max}-t}{t_{max}} + \beta_f$ $\gamma(t) = (\gamma_i - \gamma_f)\frac{t_{max}-t}{t_{max}} + \gamma_f$	$\beta_i = 2.5, \beta_f = 0.5$ $\gamma_i = 0.5, \gamma_f = 2.5$

Table 6: Adaptive coefficients strategies used in the PSO algorithm

B Hyperparameters investigation for PSO

One example of the hyperparameters investigation for the PSO algorithm to optimize the Griewank function is presented in figure 1. Each boxplot concerns the values of 10 repeated runs with a particular value for one tested hyperparameter. These are the results of the four experiments gathered in the table 2, which help to decide which combination of the hyperparameters values could lead to the best result.

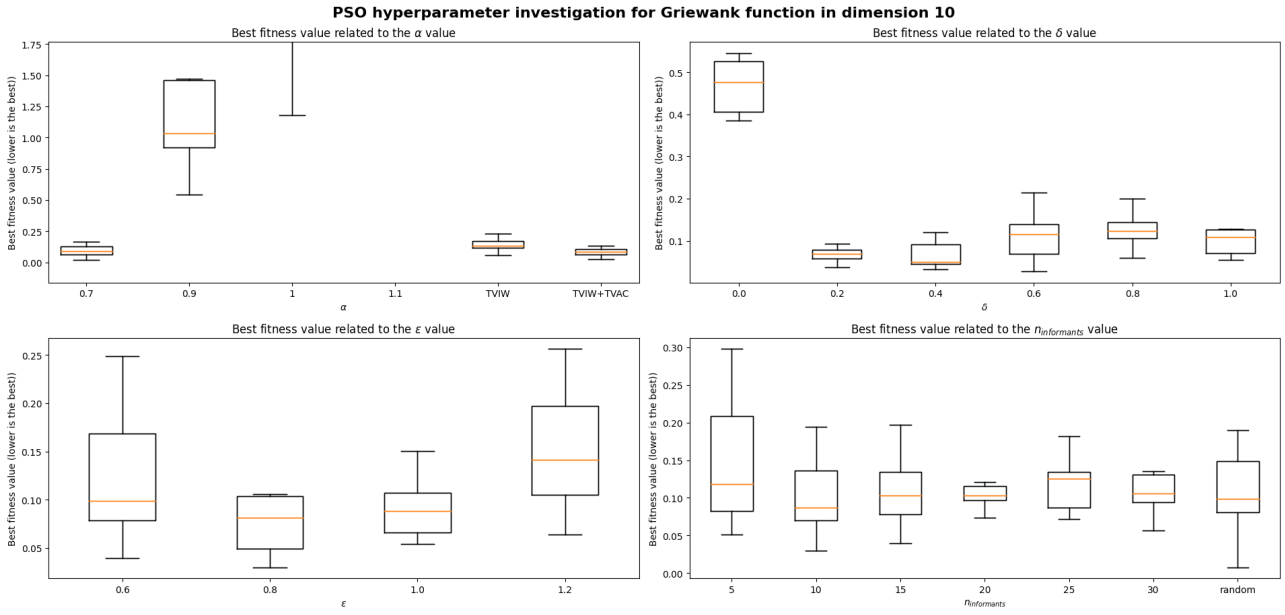

Figure 1: Hyperparameter investigation for PSO and Griewank function