Numeric function optimisation using the Particle Swarm Optimisation algorithm

Mihai Bojescu
Master in Artificial Intelligence and optimisation
Faculty of Computer Science of University "Alexandru Ioan Cuza" of Iași
Iași, Romania
bojescu.mihai@gmail.com

***Abstract*—This document contains the results of the experiments of running the Particle Swarm Optimisation algorithm on 4 benchmark functions with the scope of finding the global minumums of each of the functions. The document also details what differences do the hyperparameters of the algorithm bring in relation with the results of the numeric function minimums. The functions that the algorithms were applied on were Rastrigin's, Rosenbrock's, Michalewicz's and Griewangk's. The results were collected from 6250 runs of the algorithm using discrete values and 100 runs of the algorithm using random values, resulting in about 65GB of data. The 6250 discrete runs took about 2 hours and the 100 random values runs took about 20 minutes to finish for an Intel 1260p machine with 32GB of RAM at a constant speed of 2100MHz on all cores. The algorithm is considerably fast and moderately precise.**

***Index Terms*—PSO, Particle swarm optimisation, Rastrigin, Rosenbrock, Michalewicz, Griewangk, optimisation, numeric optimisation**

I. Introduction

Particle Swarm Optimisation is a computational method that optimises a given problem by iteratively improving upon the results of its candidate solutions with respect to the results of each solution. Its behavior is inspired from the flocking behaviour of birds, which have a natural tendence to flock towards their optimal places for food and shelter. In the algorithm, each candidate solution is represented as a particle which has its own velocity, its own personal velocity and even its own set of tunable hyperparameters.

The most significant advantage that PSO brings compared to GAs is its speed in finding the global optima, and compared to Hillclimbers is its tendency to visit the search space significantly better, thus having higher chances of finding the global optima of a given numerical function.

In this document, PSO will be used to search the 4 aforementioned functions in order to find the global minima. The used functions have well known global minimas and are highly-auditable due to being benchmark functions. The following sections will detail the used hyperparameters, the results of the algorithm, the differences the hyperparameters bring to the results and optimal hyperparameters.

II. Motivation

The motivation behind conducting this study lies in the exploration and comprehensive understanding of Particle Swarm Optimisation when applied to well-defined benchmark functions. This study focuses on elucidating how PSO behaves and adapts to different types of benchmark functions, each designed to simulate specific optimisation challenges.

III. Definitions

From this point forward, the document will use the following defintions:

- run of the algorithm = Running the algorithm $n_{functions} \cdot n_{dimensions} = 12$ times

IV. Functions

In this paper, the following functions were used in order to test the benchmark performance of the implementation:

1) Rastrigin's
2) Rosenbrock's
3) Michalewicz's
4) Griewangk's

Each of the above functions is a well-known benchmark function and has well-known global minimas, thus they can audited in a great manner.

V. Setup

A. Environment setup

The algorithm was executed on a laptop with an Intel i7 1260p processor with 12 cores running at 2100MHz and 32GB of DDR4 RAM. As for the software setup, the algorithm was run using Python 3.11.6 and numpy 1.26.0 on a system running Linux 6.1.66 LTS.

B. Paralellisation

In order to perform the computations in a more time-efficient manner, the algorithm was executed in a separate process. The process computed results for 1 set of hyperparameter and 1 function.

Executions were performed in batches of 8 in order to maximise the compute performance of the device the algorithm was run on.

C. PSO initialisation

The initialisation for the PSO algorithm's particles was performed at random with respect to the given limits:

1) For Rastringin's, $position_{initial} \in [-5.12, 5.12]^{dim}$
2) For Rosenbrock's, $position_{initial} \in [-2.048, 2.048]^{dim}$
3) For Michalewicz's, $position_{initial} \in [0, \pi]^{dim}$
4) For Griewangk's, $position_{initial} \in [-600, 600]^{dim}$

D. PSO hyperparameters

The Particle Swarm Optimisation algorithm uses a set of tunable hyperparameters:

1) w, which is the inertia weight, indicating the importance of the current direction of the current particle
2) ϕ_p, which is the cognitive parameter, indicating the tendency towards the personal best of the particle
3) r_p, which is the random cognitive parameter, which adds randomness to the cognitive parameter for each particle, with $r_p \in [0, 1]$
4) ϕ_g, which is the social parameter, indicating the tendency towards the population's best of the particle
5) r_g, which is the random social parameter, which adds randomness to the social parameter for each particle, with $r_g \in [0, 1]$
6) ϕ_r, which is the random jitter parameter, which deviates the particle from its course to hopefully find better values
7) r_r, which is the random *random* jitter parameter, which adds randomness to the random jitter parameter for each particle, with $r_r \in [0, 1]$
8) $iterations$, which is the number of

And thus, the formula for the velocity becomes:

$$\begin{aligned} v_{i,d} \leftarrow w \cdot v_{i,d} + \\ \phi_p \cdot r_p \cdot (p_{i,d} - x_{i,d}) + \\ \phi_g \cdot r_g \cdot (g_d - x_{i,d}) + \\ \phi_r \cdot r_r \cdot (random_{uniform}(0, 1)^n) \end{aligned} \tag{1}$$

Each parameter is tunable for each particle in order to increase the probability .

E. Used hyperparameters

In this paper, PSO used each combination of the following discrete values for the hyperparameters for its $5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 = 3125 \cdot 2 = 6250$ runs, generating 46GB of data:

1) $w \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$
2) $\phi_p \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$
3) $\phi_g \in \{0.0, 0.25, 0.5, 0.75, 1.0\}$
4) $\phi_r \in \{0.0, 0.25, 0.5, 1.0, 2.0\}$
5) $iterations \in \mathbb{N}, iterations = 100$

In addition to the above, a stochastic search for the optimal hyperparameters was performed, in which for 100 runs of the algorithm all of the previous parameters were picked at random from the following values, generating 19GB of data:

1) $w \in [0.0, 1.0]$
2) $\phi_p \in [0.0, 1.0]$
3) $\phi_g \in [0.0, 1.0]$
4) $\phi_r \in [0.0, 5.0]$
5) $iterations \in \mathbb{N}, iterations \in [0, 500]$

VI. Results

A. Using discrete values for the hyperparameters

For the given functions, the algorithm reached its best results in the following configurations when it used predetermined hyperparameter values:

Function	Dims.	Iters.	w	ϕ_p	ϕ_g	ϕ_r	Result
Rosenbrock	2	100	0.75	0.75	0.75	1	0.8507
	30	100	0.5	0.75	1.0	0.25	23.7119
	100	100	0.5	0.0	0.75	2	211.0306
Michalewicz	2	100	0.0	0.0	1.0	0.5	-0.7809
	30	100	0.25	0.5	0.75	1	-7.6693
	100	100	0.5	1.0	1.0	0.25	-16.5641
Griewangk	2	100	0.5	0.75	1.0	0.5	0.0681
	30	100	0.75	1.0	1.0	0.0	11.0005
	100	100	0.75	0.25	0.25	1	99.2303
Rastrigin	2	100	0.75	0.0	0.75	0.5	1.8574
	30	100	0.5	1.0	1.0	0.0	62.2359
	100	100	0.5	0.5	1.0	1.0	401.5661

Fig. 1: Best configurations and results for each function

By inspecting the results of the runs, the following table of results was constructed:

Function	Dims.	Iters.	Best	Mean	Median	Worst
Rosenbrock	2	100	0.8507	5172432488647702.0	1315.7223	1.6984463040247992e+18
	30	100	23.7119	5651452613067.534	8681.3337	3215979323453591.0
	100	100	211.0306	2814336435952.742	34693.6484	895539440289262.0
Michalewicz	2	100	-0.7809	0.0393	1.370502861939843e-44	0.9671
	30	100	-7.6693	-0.9319	-1.6650	5.5189
	100	100	-16.5641	-6.0466	-8.7856	11.3162
Griewangk	2	100	0.0681	298636717.0540	69.6293	185181230605.94724
	30	100	11.005	52585497.7302	614.3538	32767680955.6039
	100	100	99.2303	21275029.3745	2143.8349	13239089535.1134
Rastrigin	2	100	1.8574	466176982.7571	57.1042	1249342665394690.0
	30	100	62.2359	1567439093.6272	587.9396	88005482.5026
	100	100	401.5661	3537866300.1552	1849.9024	124938258.0600

Fig. 2: Results for all each function

And timings:

Function	Dims.	Best	Mean	Median	Worst
Rosenbrock	2	0.2537s	0.4898s	0.5264s	0.8967s
	30	1.027s	1.6542s	1.5225s	6.4293s
	100	2.8903s	4.4177s	3.7387s	22.6468s
Michalewicz	2	0.2793s	0.5529s	0.6123s	1.6663s
	30	1.4397s	2.5030s	2.3365s	8.8616s
	100	4.2606s	6.5478s	5.1187s	38.3012s
Griewangk	2	0.3138s	0.6627s	0.7282s	1.2361s
	30	1.5731s	2.8272s	3.0778s	17.2428s
	100	4.5282s	6.6292s	5.4489s	35.9371s
Rastrigin	2	0.2503s	0.4672s	0.5257s	0.8833s
	30	0.8120s	1.5674s	1.8188s	10.4140s
	100	2.1498s	3.5378s	3.0707s	24.3978s

Fig. 3: Timings for each function

B. Stochastic search for hyperparameters

In this approach, random values were used as hyperparameters for the PSO algorithm. Little determinism remains in the algorithm. From the 100 runs, the following best configurations and values were extracted:

Function	Dims.	Iters.	w	ϕ_p	ϕ_g	ϕ_r	Result
Rosenbrock	2	486	0.2488	0.2860	0.8344	2.2516	1.4622
	30	468	0.0587	0.8206	0.2525	4.1236	17.1632
	100	386	0.2756	0.9853	0.1875	0.0471	94.8038
Michalewicz	2	411	0.0006	0.9209	0.9351	0.9353	-0.7821
	30	162	0.6583	0.7779	0.6856	1.2427	-5.7915
	100	480	0.1479	0.2720	0.2790	1.7534	-15.0366
Griewangk	2	248	0.8295	0.3541	0.3528	2.4004	0.3774
	30	404	0.9580	0.2052	0.8905	4.7694	4.7487
	100	207	0.9121	0.1548	0.5513	1.1690	30.3857
Rastrigin	2	401	0.8133	0.7888	0.5832	3.1700	3.1526
	30	382	0.4999	0.8926	0.4431	1.6976	64.6967
	100	405	0.2188	0.6689	0.9773	3.1388	187.4847

Fig. 4: Best configurations and results for each function

Similar to the previous subsection, a table with the following parameters: best, mean, median and worst results.

Function	Dims.	Best	Mean	Median	Worst
Rosenbrock	2	1.4622	1.6474	302.7872	1.6474531444333964e+23
	30	17.1632	17351534044.6623	1249.6179	1734833610964.9421
	100	94.8038	3906404.6049	9587.0894	313951466.4197
Michalewicz	2	-0.7821	-0.0818	-1.0344732198702036e-16	0.6524
	30	-5.7915	-2.3059	-2.7970	4.2053
	100	-15.0366	-8.4531	-9.6506	6.2610
Griewangk	2	0.3774	1896448667.6981	39.3992	189644194583.7015
	30	4.7487	279511.3896	289.7916	27807028.3559
	100	30.3857	4184.1950	479.8178	244163.0585
Rastrigin	2	3.1526	289299.3240	51.0222	24180855.8853
	30	64.6967	37317.4387	525.9135	2935912.0319
	100	187.4847	1517.4707	1726.1001	3642.6197

Fig. 5: Results for all each function

Finally, timings:

Function	Dims.	Best	Mean	Median	Worst
Rosenbrock	2	0.2898s	1.0112s	0.9163s	2.6347s
	30	1.2425s	4.9661s	5.0002s	10.7783s
	100	3.6067s	12.7077s	12.4873s	29.4881s
Michalewicz	2	0.3663s	1.2831s	1.0812s	3.6374s
	30	1.9223s	7.0661s	6.5579s	16.1011s
	100	5.1177s	21.6275s	19.5296s	48.1029s
Griewangk	2	0.3287s	1.4607s	1.3627s	3.0342s
	30	1.6164s	7.4485s	7.4466s	15.6951s
	100	7.2799s	21.5996s	20.5704s	48.4895s
Rastrigin	2	0.2834s	1.1082s	1.0721s	2.4293s
	30	2.2084s	10.5571s	10.0289s	25.3332s
	100	0.9628s	3.9553s	3.8626s	9.0846s

Fig. 6: Timings for each function

C. Worst values, skewed data

For both the discrete hyperparameters and the stochastic search for hyperparameters approach, the worst results were achieved in the following case:

1) The cognitive parameter ϕ_p being set to $0.0 \Rightarrow$ The particles continued "their movement" by not considering their previously learned steps
2) The social parameter ϕ_g being set to $0.0 \Rightarrow$ The particles continued "their movement" independent of each other

This combination would make the particles move in a random direction without taking into account their known optimas, which can be considered as a free-for-all scenario. Many runs presented one or both of the traits from above, thus skewing the data towards values far from the optimas of the functions. These outliers can be easily seen in the mean, median and worst values of the presented results, and are especially present in the discrete values approach.

D. Discrete values VS Stochastic search of values approach

As a conclusion to the subsections above, we can conclude the following:

1) Runs with random hyperparameters were overall better in finding the optimas of the 4 functions as more freedom was given to the algorithm to explore
2) Due to the iterations number also being chosen at random, the timings for the stochastic runs were more variate than the timings for the discrete values approach
3) Computing the results for the stochastic approach took less than the discrete approach, while producing comparable best-case results and *better* worst-case results
4) Due to using less runs than the discrete approach, the stochastic approach used less data

VII. Conclusions

Particle swarm optimisation is a rather simple to implement algorithm that is rather hard to tune to perfection. It has a higher chance of finding global optimas, or at least values close to them. Computation-wise, the algorithm is moderately fast while being rather precise.

From the results of the runs the following could be observed: systems that worked closer to each other by having closer ties to its peers generally worked well, but so can systems that use each individuals' best performance instead of cooperation. The two strategies are not fully mutually exclusive even though there are extremes. Usually this can be observed per function basis. This further increases the sentiment that the algorithm cannot use a predefined set of hyperparameters that work well for all functions, as they need to be tuned per function basis.

The addition of a random jitter hyperparameter could also bring some benefits to the results of the function. In some instances, the algorithm could find better results using this technique of random exploration.

References

[1] Luchian Henri; Eugen Croitoru. "Genetic algorithms (2022)", Faculty of Computer Science of University "Alexandru Ioan Cuza" of Iași.
[2] https://profs.info.uaic.ro/~eugennc/teaching/nim/ (2023)
[3] Hartmut Pohlheim, The Genetic and Evolutionary Algorithm Toolbox (2006), http://www.geatbx.com/docu/fcnindex-01.html#P129_5426
[4] Hartmut Pohlheim, The Genetic and Evolutionary Algorithm Toolbox (2006), http://www.geatbx.com/docu/fcnindex-01.html#P140_6155
[5] Hartmut Pohlheim, The Genetic and Evolutionary Algorithm Toolbox (2006), http://www.geatbx.com/docu/fcnindex-01.html#P160_7291
[6] Hartmut Pohlheim, The Genetic and Evolutionary Algorithm Toolbox (2006), http://www.geatbx.com/docu/fcnindex-01.html#P204_10395
[7] https://en.wikipedia.org/wiki/Particle_swarm_optimization (2023)
[8] https://www.marksmannet.com/RobertMarks/Classes/ENGR5358/Papers/pso_bySHI.pdf (2004)
[9] https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimizati (2023)
[10] https://www.youtube.com/watch?v=JhgDMAm-imI (2018)