Bio-Inspired Computing Final Report

Testing variants of Genetic Algorithm/ Particle Swarm Optimization / Coyote Optimization Algorithm on two Multimodal benchmark functions

A1105111 王騏

1. **Overview**:

In this experiment *Genetic Algorithm* and *Particle Swarm Optimization* have been chosen as a different variation of the original metaheuristic. The three algorithms used and their significance will be discussed first, then the two multimodal benchmarks used for testing will be mentioned, lastly, the experimental setup and results will be presented. All programs and code have been written in Python with *numpy* and *matplotlib*.

2. **Metaheuristics**:

a. *Micro-Variation Chaotic Genetic Algorithm*:

Starting with the selection of the parents, since the roulette wheel could potentially get stuck at local optimum, hence a chaotic micro-variation version has been introduced as the following:

$$\alpha_{n+1} = 4 \cdot \alpha_n (1 - \alpha_n), \quad n = 1,2,3...$$

$$\beta_i = (a - b) \cdot \alpha_n$$

$$X_n^{'} = (1 - \delta) \cdot X_n + \delta \cdot \beta_i$$

where α_1 is a random number [0, 1], β is the disturbance factor and δ is the degradation factor set to 0.2. Essentially after we choose an individual, there is a probability $P_m^{'}$ =0.001that the individual chosen would be a micro-variation of itself.

Next for the crossover the paper proposed an adaptive way of setting the probability of the crossover:

$$P_c = \begin{cases} P_{c\min} - \dfrac{P_{c\max} - P_{c\min}}{1 + e^{-t}} \cdot \dfrac{f - \overline{f}}{f_{\max} - \overline{f}} & f \geq \overline{f} \\ P_{c\max} & f < \overline{f} \end{cases}$$

where f is the maximum fitness of the chosen parent pair, $\overline{f}$ is the mean fitness of the population, $f_{\max}$ is the maximum fitness of the population and that $P_{c\max} = 0.8$， $P_{c\min} = 0.5$. In other words if the fitness is better than the mean fitness, then the probability of crossover increases.

The crossover itself is identical to the original:

$$X_n^{'} = \gamma \cdot X_n + (1-\gamma) \cdot X_{n+1}$$
$$X_{n+1}^{'} = \gamma \cdot X_{n+1} + (1-\gamma) \cdot X_n$$

where γ is a random number [0, 1]

Mutation is pretty similar to the crossover where a sigmoid like function has been introduced:

$$P_m = \begin{cases} P_{m\max} - \dfrac{P_{m\max} - P_{m\min}}{1+e^t} \cdot \dfrac{f - \overline{f}}{f_{m\max} - \overline{f}} & f \geq \overline{f} \\ P_{m\max} & f < \overline{f} \end{cases}$$

$$x_i^{'} = (\frac{a-b}{2}) + (a-b) \cdot (\varphi - 0.5)$$

where φ is a random number [0, 1], a, b are upper and lower bounds and the f here is the fitness of the current individual that is being evaluated.

One thing to notice is that the exponential in both crossover and mutation could be astronomically large or small after t = 20, hence a function is added to the implementation to prevent such a phenomenon.

Before going on to the next iteration, a chaos perturbation is introduced as:

$$X_n^{'} = X_n \cdot \eta + \alpha_{n+1} \cdot (a - X_n) \cdot (1-\eta)$$

where η is a random number [0, 1]. The alpha is the same chaotic map used early in the selection stage.

b. *Adaptive Weight Particle Swarm Optimization*:

Here's a reminder of the original PSO equation:`

$$\begin{aligned} v_i(k+1) &= w \times v_i(k) + c_1 \times r_1 \times (p_i(k) - x_i(k)) \\ &\quad + c_2 \times r_2 \times (p_g(k) - x_i(k)) \\ x_i(k+1) &= x_i(k) + v_i(k+1) \end{aligned}$$

where c1 and c2 are fixed arbitrary constants.
The weight is an adaptive weight:

$$w = w_1 - (w_1 - w_2) \times \frac{k}{maxiter}$$

where w1 = 0.9, w2 = 0.4, and k is the kth iteration.

Since the velocity of an individual particle gets accelerated according to the distances from the particle to its pbest and gbest, the paper proposed an adaptive weighting strategy that adaptively controls the two coefficients with respect to pbest and gbest:

$$v_i(k+1) = w \times v_i(k) + c_{g_{pi}}(k) \times r_1 \times g_{pi}(k) + c_{g_{gi}}(k) \times r_2 \times g_{gi}(k)$$
$$x_i(k+1) = x_i(k) + v_i(k+1)$$

And the following:

$$c_{g_{pi}}(k) = F(g_{pi}(k))$$
$$c_{g_{gi}}(k) = F(g_{gi}(k))$$

where $g_{pi}(k)$ and $g_{gi}(k)$ are defined by:

$$g_{pi}(k) = p_i(k) - x_i(k)$$
$$g_{gi}(k) = p_g(k) - x_i(k),$$

which denotes the Euclidean distances of particle i from its pbest and gbest. And F(.) represents the adaptive weighting function:

$$F(D) = \frac{b}{1 + e^{-a \times (D-c)}} + d$$

It has been suggested by the paper that a = 0.000035 * search_range, b = 0.5, c = 0, d = 1.5. D would be the Euclidean distance calculated.

c. *Coyote Optimization Algorithm*:

Initialization starts as the following:

$$soc_c^{p,t} = \vec{x} = (x_1, x_2, ..., x_D)$$

where $soc_{c,j}^{p,t}$ is the social condition of the pth pack, tth iteration, jth dimension of the cth coyote initialized as:

$$soc_{c,j}^{p,t} = lb_j + r_j \cdot (ub_j - lb_j)$$

Social conditions are updated as:

$$new\_soc_c^{p,t} = soc_c^{p,t} + r_1 \cdot \delta_1 + r_2 \cdot \delta_2$$

$$\delta_1 = alpha^{p,t} - soc_{cr_1}^{p,t}, \quad \delta_2 = cult^{p,t} - soc_{cr_2}^{p,t}$$

where δ_1 and δ_2 are the alpha and social influences respectively, and both cr1 and cr2 are random coyotes in the pack.

The $alpha^{p,t}$ is the coyote with the best fitness of the pth pack,

$$alpha^{p,t} = \{soc_c^{p,t} | arg_{c=\{1,2,\ldots,N_c\}} min f(soc_c^{p,t})\}$$

and $cult_j^{p,t}$ is the median of the pth pack based on the ranking of fitness.

$$cult_j^{p,t} = \begin{cases} O^{p,t}_{\frac{(N_c+1)}{2},j}, & N_c \ is \ odd \\ \frac{O^{p,t}_{\frac{N_c}{2},j} + O^{p,t}_{(\frac{N_c}{2}+1),j}}{2}, & otherwise \end{cases}$$

For every pack there is a chance of newborn coyotes, and the initialization of the pup is as:

$$pup_j^{p,t} = \begin{cases} soc^{p,t}_{r_1,j}, & rnd_j < (P_s+1)/2 \\ soc^{p,t}_{r_2,j}, & rnd_j \geq (P_s-1)/2 \\ R_j, & otherwise \end{cases}$$

and that $P_s = 1/D$ is the scatter probability.
Both conditions on the top are optimized by the author and different from the original conditions that were proposed in the paper.
In addition, the conditions that the pup survives are:

if $\varphi = 1$ **then**
The pup survives and the only coyote in ω dies.
elseif $\varphi > 1$ **then**
The pup survives and the oldest coyote in ω dies.
else
The pup dies.

Afterwards, for every iteration there is a chance that a coyote would leave its pack. For the purpose of generalizing the pack sizes, two coyotes would swap packs if the conditions are met:

$$P_e = 0.005 \cdot N_c^2$$

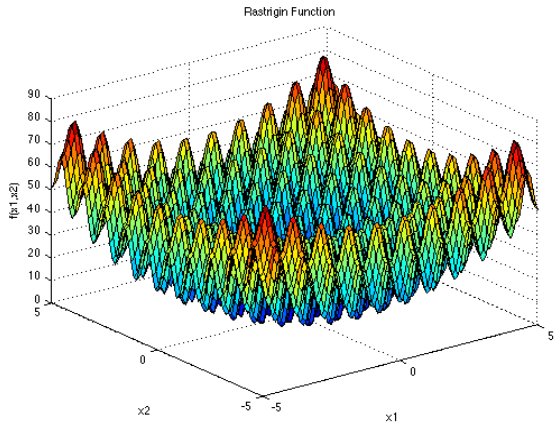
Since the probability could be over 1 if $Nc > \sqrt{200}$, so the population size should be less than 200.

3. **Benchmark Functions**:

a. *Rastrigin*:

$$f(\mathbf{x}) = 10n_{var} + \sum_{i=1}^{v_{var}} \left(x_i^2 - 10\cos(2\pi x_i)\right)$$

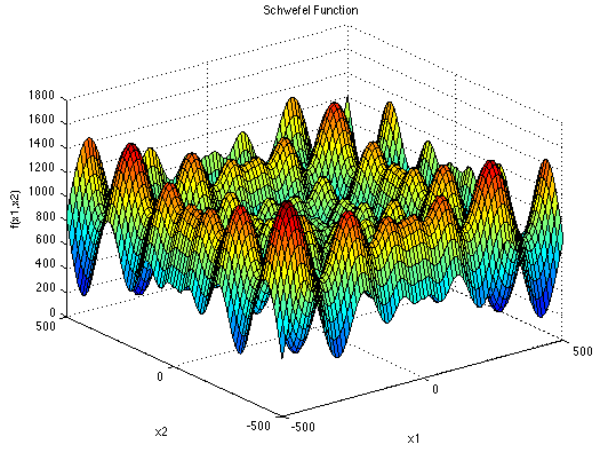
Global optimum at $\mathbf{x}^* = [0 \quad 0 \ldots 0]$ and $f(\mathbf{x}^*) = 0$
with search space $\mathbf{x} \in [-5.12, \ 5.12]$in the hypercube.



b. *Schwefel 2.26*:

$$f(\mathbf{x}) = 418.9829 n_{var} - \sum_{i=1}^{n_{var}} x_i \sin(\sqrt{|\mathbf{x}|})$$

Global optimum at $\mathbf{x}^* = 420.9687[1,\ 1, \ldots 1]$ and $f(\mathbf{x}^*) = 0$ with search space $\mathbf{x} \in [-500,\ 500]$in the hypercube.



4. **Experimental Setup**:

For the population size, since *PSO* generally performs well when nPop = 30, *COA* and *GA* are usually tested with nPop = 100, therefore the population size selected is nPop = 70.

Dimensions are set to the required d = 6 and the search ranges are given by the benchmark functions above.

Stopping criteria is set to iterMax = 5000 and number experiments is set to nExp = 30.

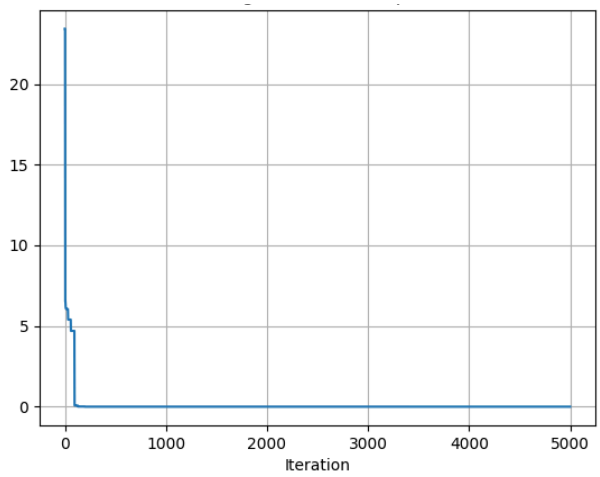
5. **Experimental Results**:

gBest is the global optimum found and iterBest is the iteration when the actual global optimum is found. Error margin is set to 1e-6 so if gBest is less than the margin then it counts as successful. The best experimental results are based on first the gBest found then the iteration when it is found.

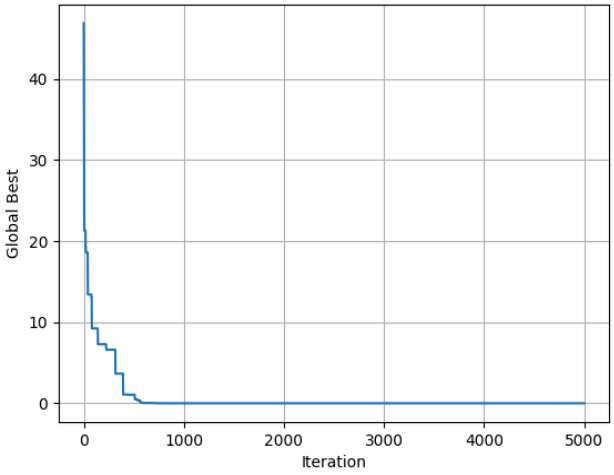
Rastrigin:

	MVCGA	*AWPSO*	*COA*
gBest minimum	0.0	0.0	0.0
gBest maximum	3.3638885e-06	0.0	0.0
gBest average	1.3473631e-07	0.0	0.0
gBest median	5.5145576e-10	0.0	0.0
gBest std deviation	6.0195401e-07	0.0	0.0
iterBest minimum	3162	1391	755
iterBest maximum	5000+	1771	1038
iterBest average	4938.73	1511.93	892.06
iterBest median	5000.0+	1506.5	903.0
iterBest std deviation	329.93	73.67	65.58
Success rate	29 / 30 = 96.67 %	30 / 30 = 100.00 %	30 / 30 = 100.00 %
Average runtime (s)	23.67	0.27	2.46

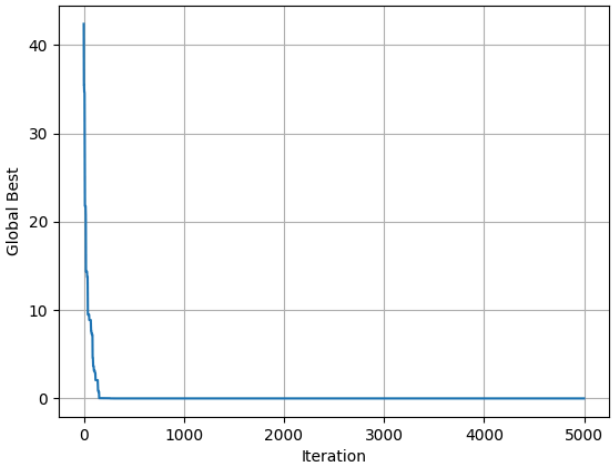
Best results convergence curve:



MVCGA	
gBest	0.0
iterBest	3162
Runtime (s)	15.62



AWPSO	
gBest	0.0
iterBest	1391
Runtime (s)	0.18

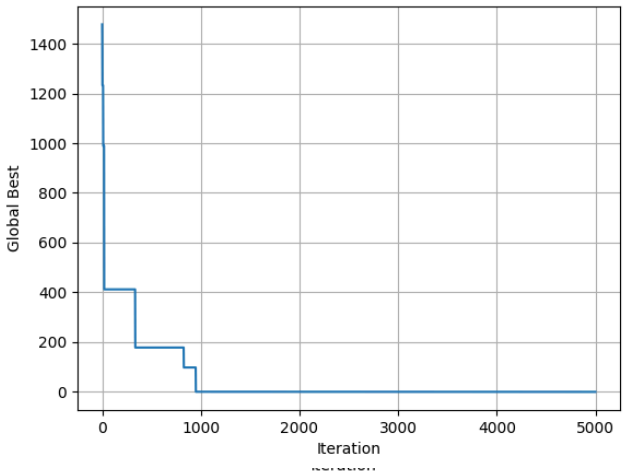


COA	
gBest	0.0
iterBest	755
Runtime (s)	2.09

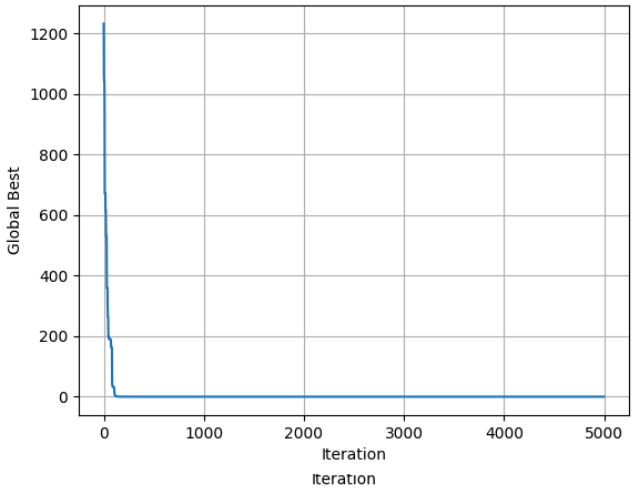
Schwefel 2.26:

	MVCGA	*AWPSO*	*COA*
gBest minimum	7.6365397e-05	7.6365397e-05	7.6365397e-05
gBest maximum	2.5278608	712.1471435	7.6365397e-05
gBest average	2.1639330e-01	272.5093832	7.6365397e-05
gBest median	1.5453330e-03	236.8767455	7.6365397e-05
gBest std deviation	5.4009320e-01	178.8549890	7.6365397e-05
iterBest minimum	5000+	5000+	5000+
iterBest maximum	5000+	5000+	5000+
iterBest average	5000.0+	5000.0+	5000.0+
iterBest median	5000.0+	5000.0+	5000.0+
iterBest std deviation	0.0	0.0	0.0
Success rate	0 / 30 = 0.00 %	0 / 30 = 0.00 %	0 / 30 = 0.00 %
Average runtime (s)	23.27	0.61	12.7513

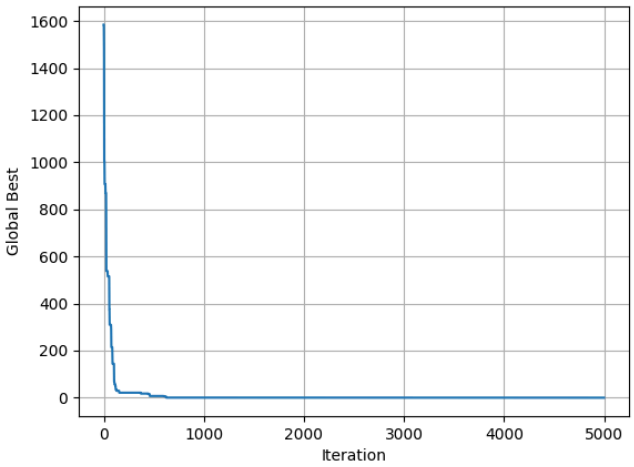
Best results convergence curve:



MVCGA	
gBest	7.6365397e-05
iterBest	5000+
Runtime (s)	22.78



AWPSO	
gBest	7.6365397e-05
iterBest	5000+
Runtime (s)	0.59



COA	
gBest	7.6365397e-05
iterBest	5000+
Runtime (s)	11.97

6. **Conclusion**:

For *Rastrigin*, it's clear that all three algorithms could all find the desired global optimum. In terms of computational time, *AWPSO* is the clear winner here with an average runtime of just 0.27 seconds. Both *AWPSO* and *COA* have a success rate of 100% in all 30 experiments, which shows the consistency when solving the first benchmark function. *MVCGA* however, is much slower than the other two and fails to find the optimum multiple times despite being very close to it.

Evidently, *Schwefel* is a much tougher objective function to solve than the first. Out of all 90 experiments, there were a total of 0 successful attempts in finding the optimum. What is interesting is that the best results from all three algorithms land at the value 7.6365397e-05, with *COA* being the most stable with a consistency of 100% and *AWPSO* again with the fastest average time. However the results of *AWPSO* are all over the place with a standard deviation of 178.8549890. *MVCGA* on the other hand is also pretty consistent, but the results are worse than what *COA* could produce.

From the results shown above, it is very straightforward that *AWPSO* is a very fast algorithm, though it fails to perform when the FOBJ becomes more complicated. Both *COA* and *MVCGA* are very robust and steady with *COA* being better in both computational time and the ability to search for the optimum.

Obviously the algorithms are implemented based on the papers and could very much have slight errors and mistakes that affects how the results are outputted. In addition, as to why the best values found for *Schwefel* are the same across the board, I believe it could be a couple of reasons. Incomplete or incorrect implementation of the metaheuristics, the values of the variables that were set up for the experiment and so on.

Nonetheless, I have learned a lot in this experiment while implementing the algorithms and fine tuning some of the variables. I think the most difficult part of the whole process was filling the holes where the papers didn't mention. It was inherently challenging when trying to put the missing puzzle together to make the flow of the algorithms more logical. Regardless, it has been a fun and memorable experience.

7. **References**:

Liu, W., Wang, Z., Yuan, Y., Zeng, N., Hone, K., & Liu, X. (2021). A novel s sigmoid-function-based adaptive weighted particle swarm optimizer. *IEEE Transactions on Cybernetics, 51*(2), 1085–1093. [1]

Pierezan, J., & Coelho, L. S. (2018). Coyote optimization algorithm: A new metaheuristic for global optimization problems. *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2633–2640. [2]

Pan, W., Yan, M., Tan, Y., & Xie, H. (2018). Micro variation chaos genetic algorithm. In *Proceedings of the 2018 Chinese Control and Decision Conference (CCDC)* (pp. 4523–4528). IEEE. [3]

**NOTE:

If you would like an easier way to run and test the program through a main file, I have it on my github account. It contains code that runs all three algorithms and uses the two multimodal functions.