Impact Factor 6.1

Journal of Cyber Security

ISSN:2096-1146

Scopus

DOI

Google Scholar

More Information
www.journalcybersecurity.com

Crossref doi Google Scholar scopus

From Poetry to a Metaheuristic: Designing the Shakespearean Soliloquy Optimization (SSO) Algorithm

Mitat Uysal
Dogus University,Istanbul,Turkey

S.Aynur Uysal
Dogus University,Istanbul,Turkey

Abstract- Many metaheuristic algorithms are created by taking inspiration from nature, society, art, or human behavior. However, sometimes these ideas look new only because of the story behind them. If the mapping between the story and the mathematical search process is not clear, measurable, and repeatable, the method may be misleading.

In this paper, based on basic principles of stochastic search, population behavior, and standard benchmark testing, we do two things. First, we propose a general guideline that shows how a metaheuristic algorithm can be systematically built from a poem stanza or from the known traits of a famous person. Second, we introduce a new algorithm called Shakespearean Soliloquy Optimization **(SSO)**. This method is inspired by dramatic elements in Shakespeare's works such as soliloquy (self-reflection), rhythmic movement, chorus interaction, and the conflict–resolution story arc.

We test SSO on a simple quadratic function $z = (x-5)^2 + (y-4)^2$and compare its performance with PSO, ABC, and a continuous version of MBO. Then we evaluate it on five well-known benchmark functions: Sphere, Rosenbrock, Rastrigin, Ackley, and Griewank.

Keywords- Metaheuristics, metaphor-to-algorithm mapping, Shakespearean soliloquy, swarm intelligence,

1. PROBLEM STATEMENT AND NOTATION

We consider unconstrained minimization:

$$\min_{\mathbf{x}\in\Omega\subset\mathbb{R}^d} f(\mathbf{x}), \qquad \Omega = \{\mathbf{x}: \ell_j \le x_j \le u_j, j = 1, \dots, d\}.$$

A population-based metaheuristic maintains N candidate solutions $\{\mathbf{x}_i\}_{i=1}^{N}$and iteratively updates them:

$$\mathbf{x}_i^{t+1} = \mathcal{U}\left(\mathbf{x}_i^t;\ \Theta_t, \text{population statistics}\right),$$

with boundary handling (clipping):

$$x_{i,j}^{t+1} \leftarrow \min\{u_j, \max\{\ell_j, x_{i,j}^{t+1}\}\}.$$

where $U(\cdot)$is the update operator, $\Theta_t$denotes the algorithm control parameters at iteration t (e.g., step size, randomness coefficients, or weights), and S_t represents population statistics (such as mean position, best solution, or diversity measures).

2. HOW TO "GENERATE" A METAHEURISTIC FROM POETRY OR PERSONALITY

Many metaphor-based algorithms fail because the metaphor is not tied to verifiable operators (mutation, recombination, learning, selection) and ends up re-labeling existing moves [1]. To avoid that, we propose a 5-step construction protocol:

Step A — Extract "Mechanisms"
From poem/personality, extract actionable mechanisms:

- *Self-reflection* → local improvement around the current point (intensification).
- *Chorus / crowd influence* → movement toward elite/consensus points (social learning).
- *Conflict / sudden plot twist* → occasional long jumps (diversification).
- *Rhythm / meter* → periodic alternation of exploration vs exploitation schedules.

Step B — Map each mechanism to a canonical operator
Use known operator templates (all standard in stochastic search literature [2-4]):

- *Attraction to best*: $x \leftarrow x + \alpha(g - x)$
- *Peer difference*: $x \leftarrow x + \beta(x_r - x_s)$
- *Gaussian / Lévy-like jump (here: Gaussian):* $x \leftarrow x + \sigma\epsilon, \epsilon \sim \mathcal{N}(0, I)$
- *Selection: accept-if-better (elitist replacement)*

Step C — Add a measurable exploration–exploitation schedule
Define $\rho(t) \in [0,1]$controlling how often you do local vs global moves. Schedules can be linear, cosine, or adaptive.

Step D — Ensure "No Free Lunch" awareness
Do not claim universal superiority; use a diverse test set and report runtime + accuracy [5].

Step E — Benchmark with transparent baselines
Compare to canonical optimizers (e.g., PSO [6], ABC [7–9], MBO [10]) and report identical budgets.

3. SHAKESPEAREAN SOLILOQUY OPTIMIZATION (SSO)

3.1 Literary inspiration

We operationalize Shakespeare's dramatic devices as four update operators:

1. *Soliloquy (self-dialogue)***:** intensify around personal best p_i
2. *Chorus (collective voice)***:** attract toward global best g
3. *Iambic rhythm:* alternate (or smoothly blend) exploration/exploitation over time
4. *Tragic twist:* rare disruptive jump to escape local minima

This keeps the metaphor, but the algorithm is fully mathematical and reproducible (a key critique point in [9]).

3.2 SSO equations (continuous, dimension d)

Maintain personal bests $p_i$and global best g.

Rhythm schedule (smooth iambic alternation):

$$r(t) = \frac{1}{2}\left(1 + \cos\left(2\pi\frac{t}{T}\right)\right) \in [0,1]$$

where $T$is max iterations. High $r(t)$emphasizes exploitation.

Soliloquy step (local refinement):

$$s_i^t = x_i^t + \alpha(t)\,(p_i^t - x_i^t) + \eta(t)\,\epsilon_i^t, \epsilon \sim \mathcal{N}(0, I).$$

Chorus step (social learning):
Pick two random indices $a \neq b \neq i$:

$$\mathbf{c}_i^t = \mathbf{x}_i^t + \beta(t)\,(\mathbf{g}^t - \mathbf{x}_i^t) + \gamma(t)\,(\mathbf{x}_a^t - \mathbf{x}_b^t).$$

Tragic twist (rare jump):
With probability $p_{\text{twist}}(t)$:

$$\mathbf{j}_i^t = \mathbf{x}_i^t + \delta(t)\,\mathbf{u}, u_j \sim \mathcal{U}[-1,1].$$

Rhythm blend + selection (elitist):
Construct candidate:

$$\mathbf{y}_i^t = r(t)\,\mathbf{s}_i^t + (1 - r(t))\,\mathbf{c}_i^t,$$

then if twist triggers, replace $\mathbf{y}_i^t \leftarrow \mathbf{j}_i^t$.
Finally accept-if-better:

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{y}_i^t & \text{if } f(\mathbf{y}_i^t) < f(\mathbf{x}_i^t), \\ \mathbf{x}_i^t & \text{otherwise.} \end{cases}$$

Update $\mathbf{p}_i, \mathbf{g}$ accordingly.

Parameter schedules (typical):

$$\alpha(t) = \alpha_0\left(1 - \frac{t}{T}\right), \beta(t) = \beta_0, \eta(t) = \eta_0\left(1 - \frac{t}{T}\right), \delta(t) = \delta_0\left(1 - \frac{t}{T}\right).$$

and $p_{\text{twist}}(t) = p_0\left(1 - \frac{t}{T}\right)$.

4. BASELINES (for fair comparison)

4.1 PSO (canonical form)

Particle Swarm Optimization (PSO) is a population-based stochastic search method where each candidate solution, called a particle, moves in the search space with a velocity influenced by its own best experience and the global best solution found by the swarm. The movement balances exploration and exploitation through random weighting factors [6].

$$\mathbf{v}_i^{t+1} = \omega\mathbf{v}_i^t + c_1 r_1(\mathbf{p}_i - \mathbf{x}_i^t) + c_2 r_2(\mathbf{g} - \mathbf{x}_i^t), r_1, r_2 \sim \mathcal{U}[0,1]$$
$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}.$$

4.2 ABC (continuous form)

Artificial Bee Colony (ABC) models the foraging behavior of honey bees, where candidate solutions represent food sources and new neighbors are generated by local perturbations guided by random differences between solutions.

$$v_{i,j} = x_{i,j} + \phi_{i,j}\,(x_{i,j} - x_{k,j}), \phi_{i,j} \sim \mathcal{U}[-1,1].$$

Selection is greedy; onlooker selection probabilities often use normalized fitness; scouts reinitialize abandoned sources. ([7-9].)

4.3 MBO (continuous adaptation)

MBO is originally designed around V-formation information sharing and leader replacement [5]. For continuous problems, a common adaptation is: keep a leader and two "wings," propose multiple neighbor perturbations for each bird, share the best candidates backward along the formation, then rotate leader periodically. (MBO origin and concept: [10].)

5. TEST PROBLEMS AND EVALUATION SETUP

To illustrate the behavior of the algorithms on a simple convex landscape, we first consider the *Quadratic target function:* $f(x, y) = (x - 5)^2 + (y - 4)^2$.
The convergence curves are shown in Figure 1, while Figure 2 presents the trajectories of the solutions on the contour plot.
Figure 3 compares the runtime performance of the methods, and Figure 4 illustrates the 3D surface of the function together with the best solution found by SSO.

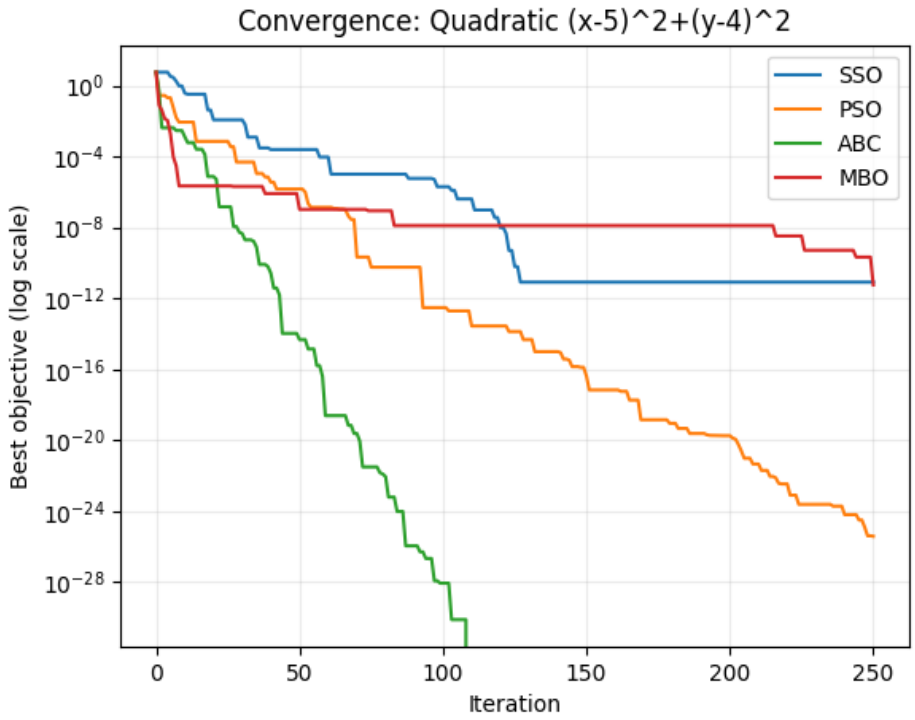

Figure-1-Convergence

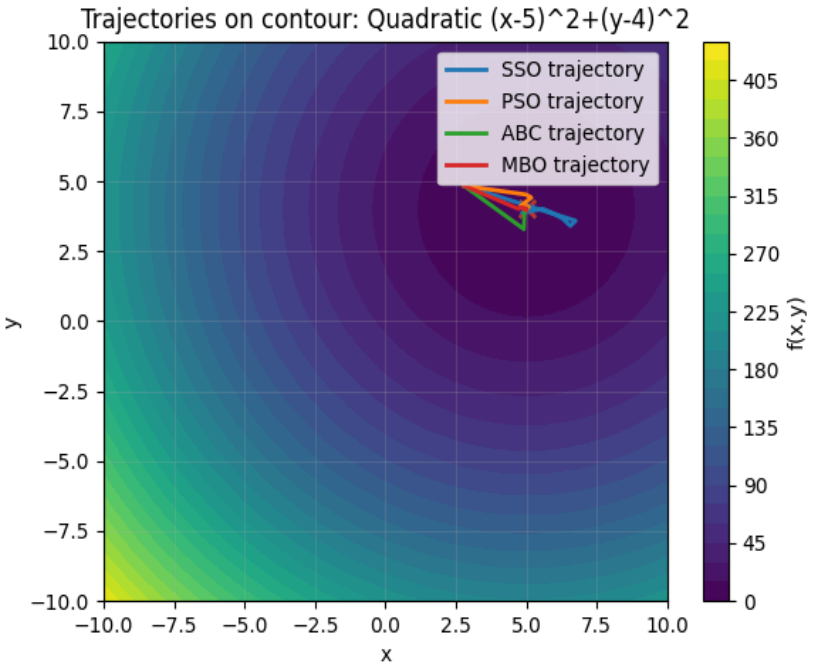

Figure-2-Trajectories on Contour

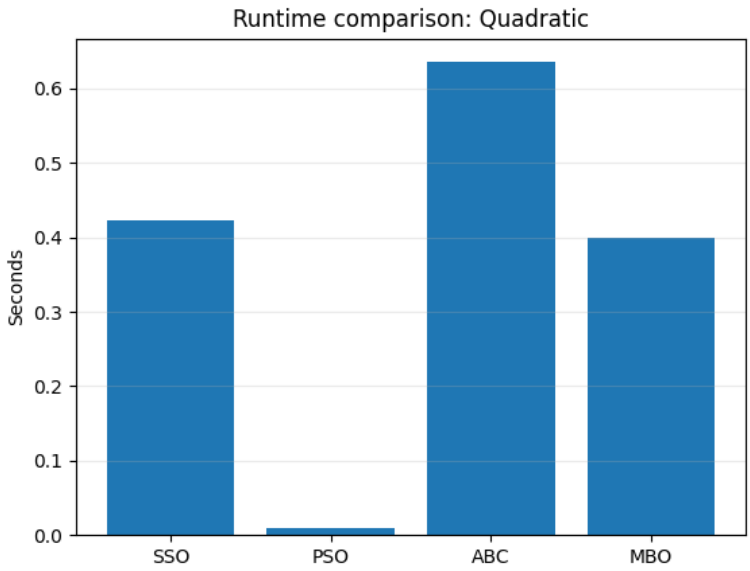

Figure-3-Runtime Comparison

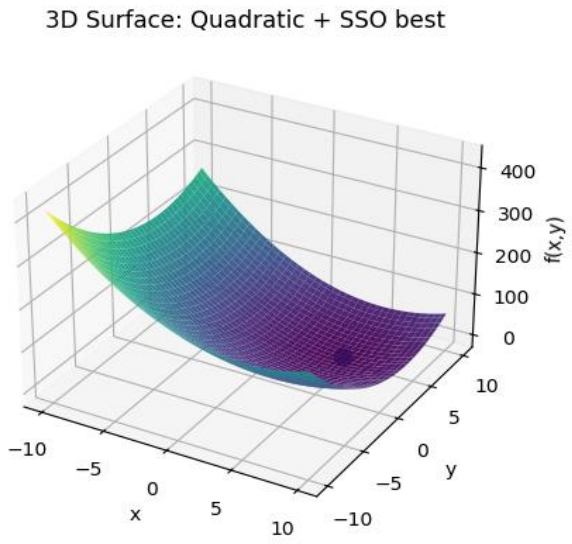

Figure-4- 3D Surface:Quadratic+SSO Best

After analyzing the quadratic test case, we further evaluate the algorithms on standard benchmark functions to assess their general performance.

1. **Sphere:** $f(\mathbf{x}) = \sum x_i^2$ (standard in benchmark surveys [2–4]).

The performance of the algorithms on the Sphere function is illustrated through convergence curves, trajectories on the contour plot, runtime comparison, and the 3D surface with the SSO best solution (Figures 5–8).

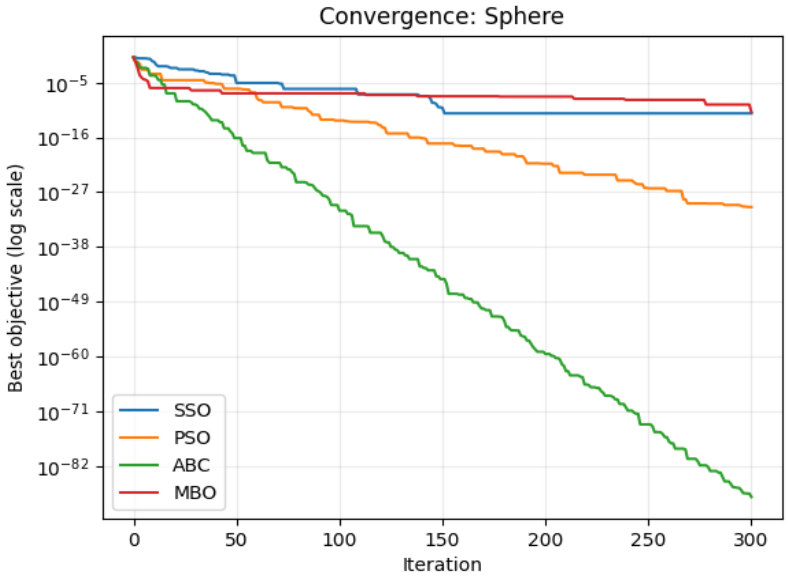

Figure-5-Convergence:Sphere

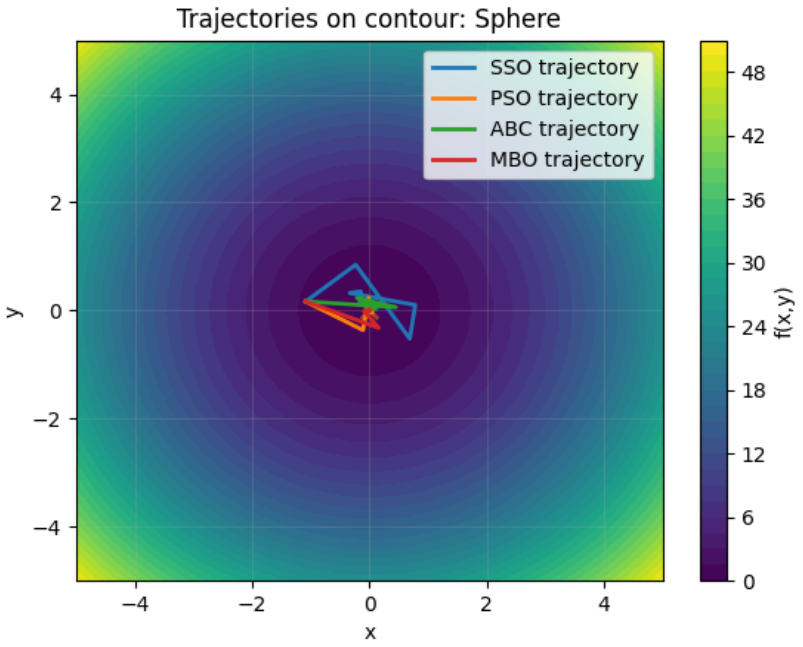

Figure-6-Trajectories on contour:Sphere

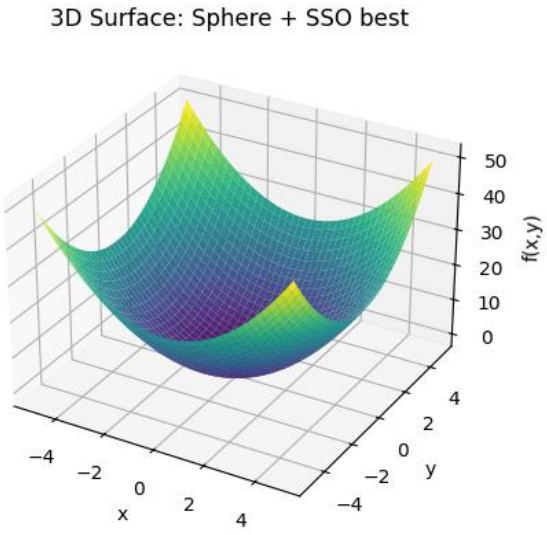

Figure-7-3D Surface:Sphere+SSO Best

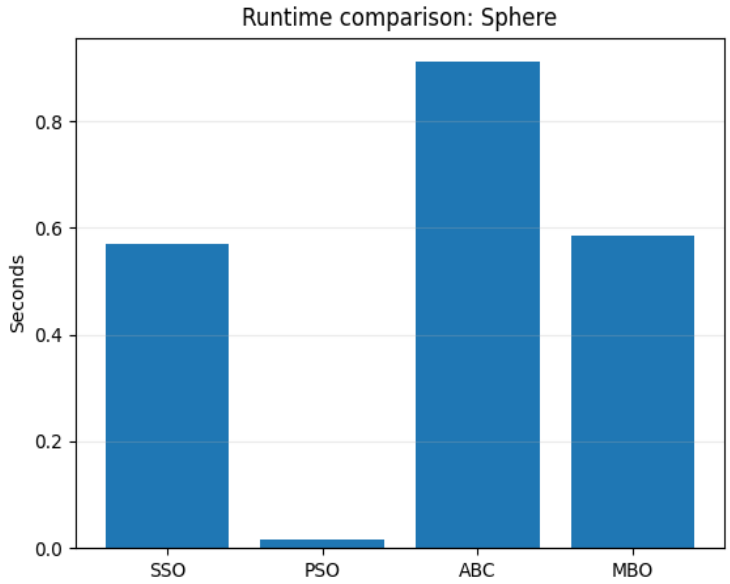

Figure-8- Runtime Comparison :Sphere

2. **Rosenbrock (banana):**

Next, we evaluate the algorithms on the Rosenbrock function, a standard curved-valley optimization test [11],[12].

$$f(x,y) = (a-x)^2 + b(y-x^2)^2, (a,b) = (1{,}100)$$

The behavior of the algorithms on the Rosenbrock function is shown through convergence curves, contour trajectories, runtime comparison, and the 3D surface with the SSO best solution (Figures 9–12).

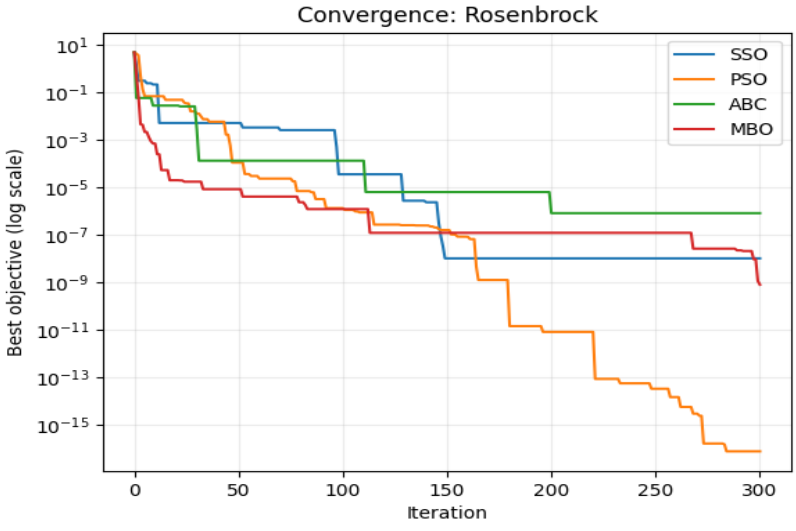

Figure-9-Convergence:Rosenbrock

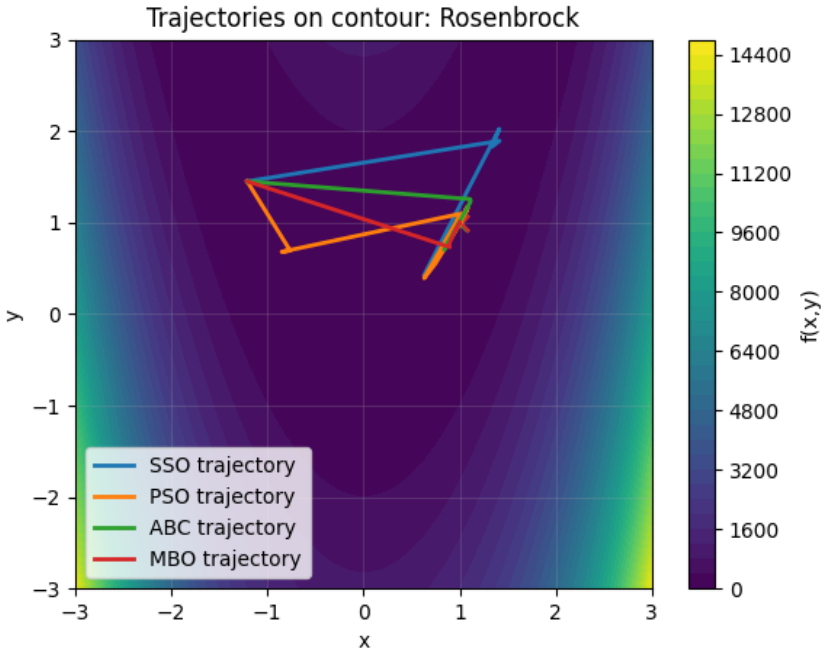

Figure-10-Trajectories on Contour:Rosenbrock

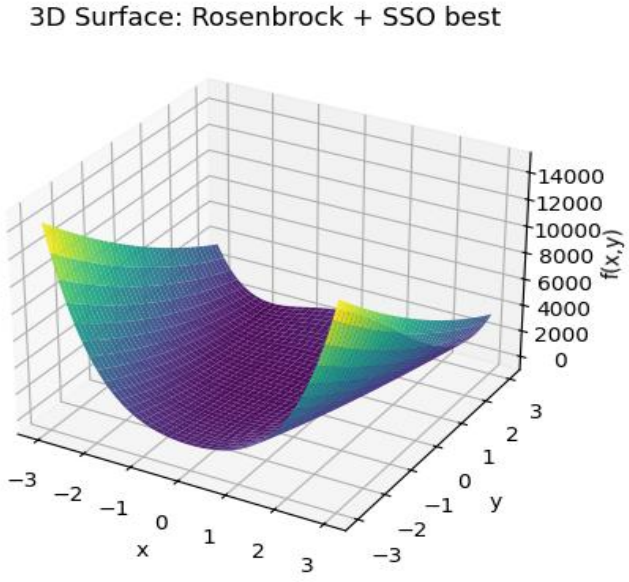

Figure-11-3D Surface :Rosenbrock+SSO Best

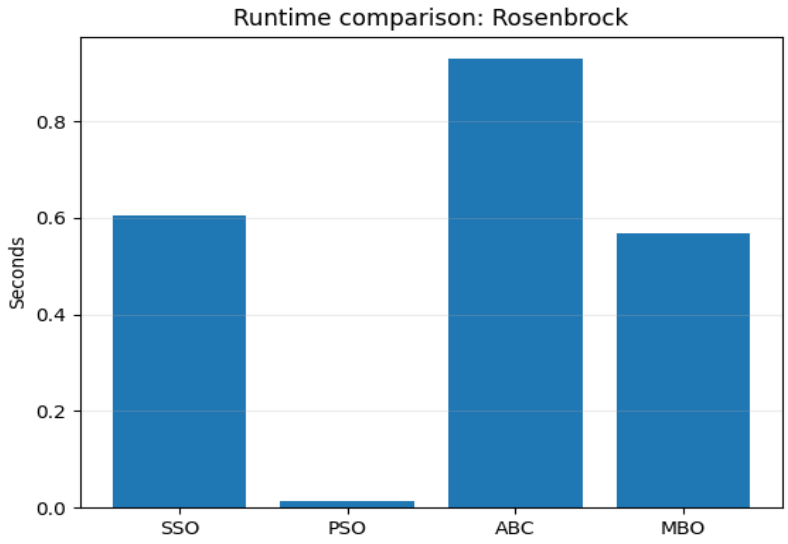

Figure-12-Runtime Comparison:Rosenbrock

3. **Rastrigin:**

To examine performance on a highly multimodal landscape with many local minima, we next use the Rastrigin function [13].

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} (x_i^2 - A\cos(2\pi x_i)), A = 10$$

Results for the Rastrigin function are presented in Figures 13–16.

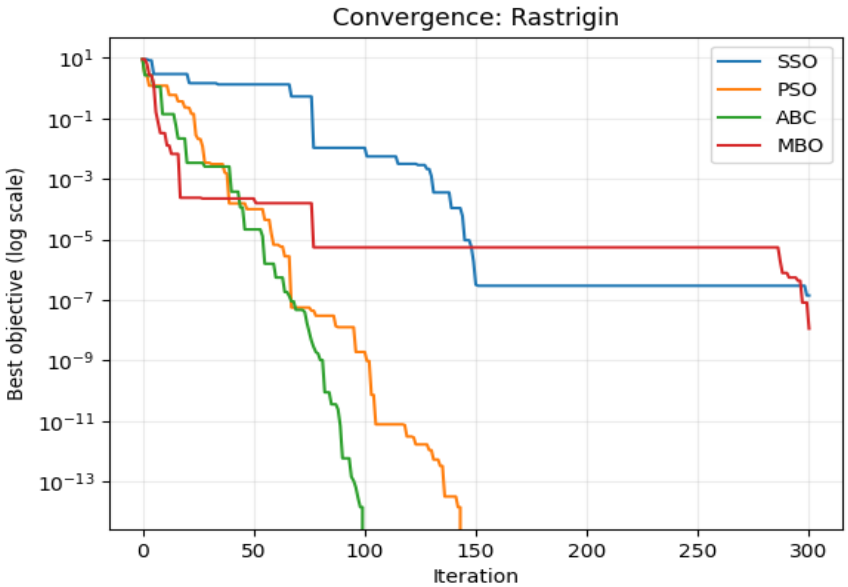

Figure-13-Convergence:Rastrigin

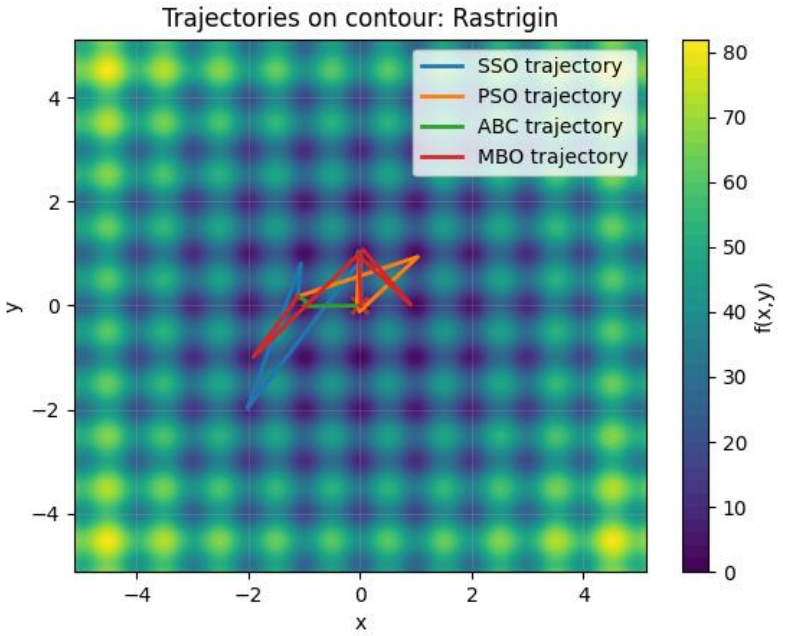

Figure-14-Trajectories on Contour:Rastrigin

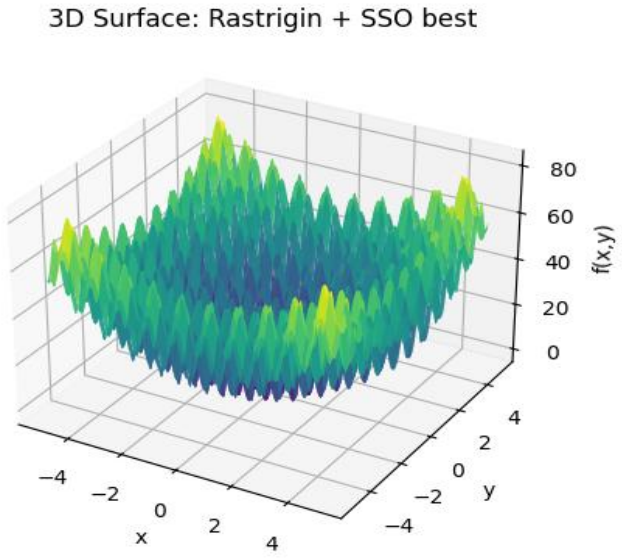

Figure-15-3D Surface:Rastrigin+SSO Best

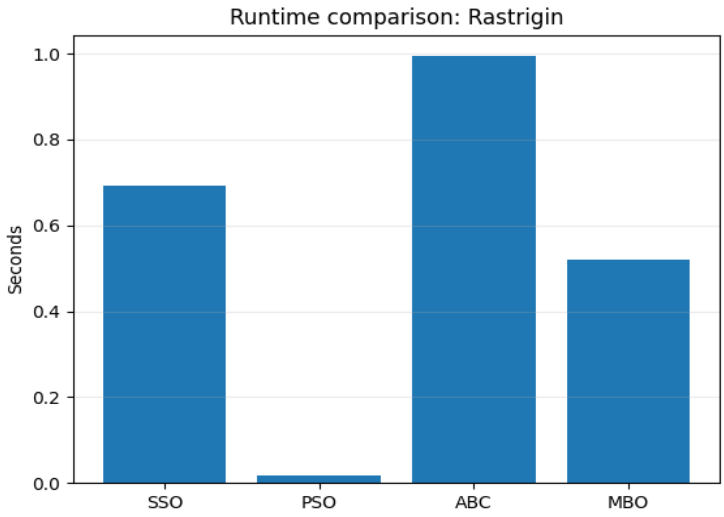

Figure-16-Runtime Comparison:Rastrigin

4. **Ackley:**

To further evaluate performance on a complex landscape with many local optima and a flat outer region, we next consider the Ackley function [14].

$$f(\mathbf{x}) = -a\exp\ \left(-b\sqrt{\frac{1}{n}\sum x_i^2}\right) - \exp\left(\frac{1}{n}\sum\cos\left(cx_i\right)\right) + a + e,$$

with $a = 20, b = 0.2, c = 2\pi$.
The results for the Ackley function are shown through convergence curves, contour trajectories, runtime comparison, and the 3D surface with the SSO best solution (Figures 17–20).

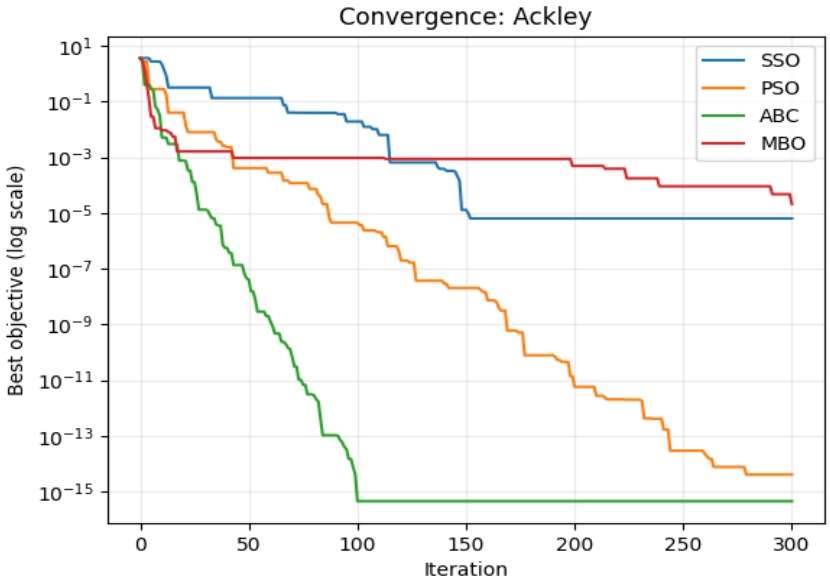

Figure-17- Convergence:Ackley

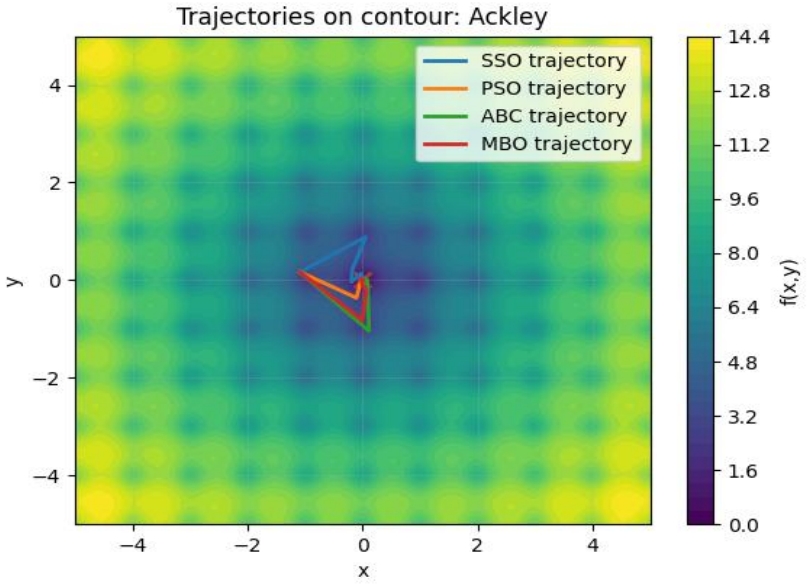

Figure-18-Trajectories on Contour:Ackley

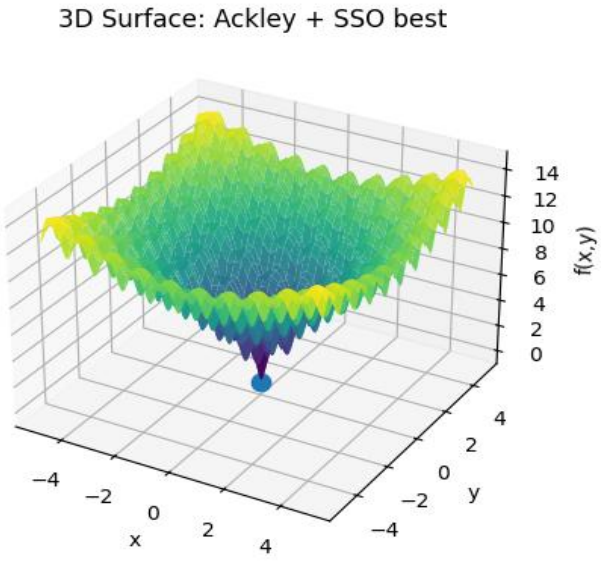

Figure-19- 3D Surface:Ackley+SSO Best

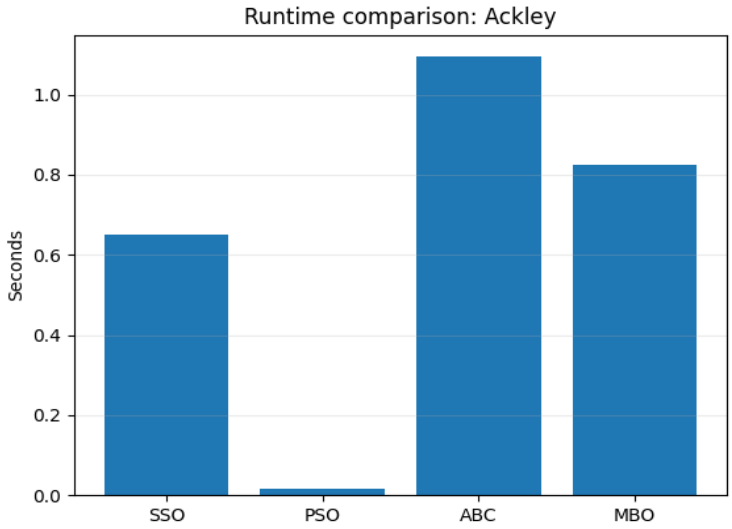

Figure-20-Runtime Comparison:Ackley

5. **Griewank:**

Finally, we evaluate the algorithms on the Griewank function, which combines many widespread local minima with a regularly structured landscape [15].

$$f(\mathbf{x}) = 1 + \frac{1}{4000}\sum x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

The performance on the Griewank function is illustrated by convergence curves, contour trajectories, runtime comparison, and the 3D surface with the SSO best solution (Figures 21-24).

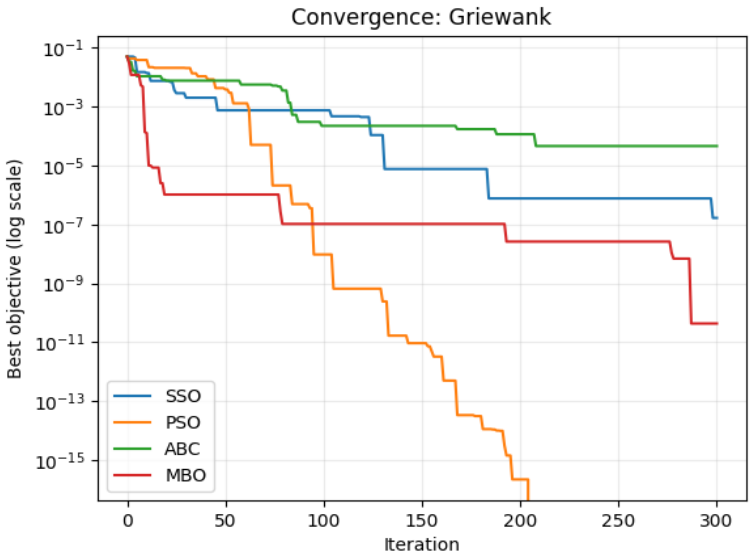

Figure-21-Convergence:Griewank

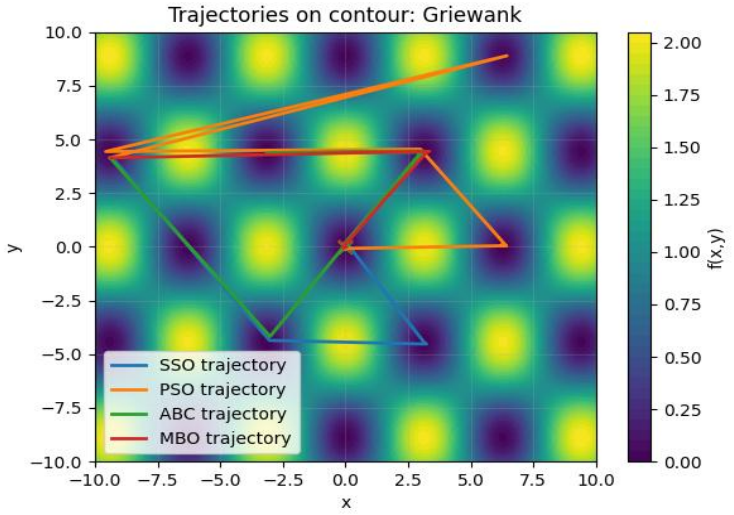

Figure-22-Trajectories on Contour:Griewank

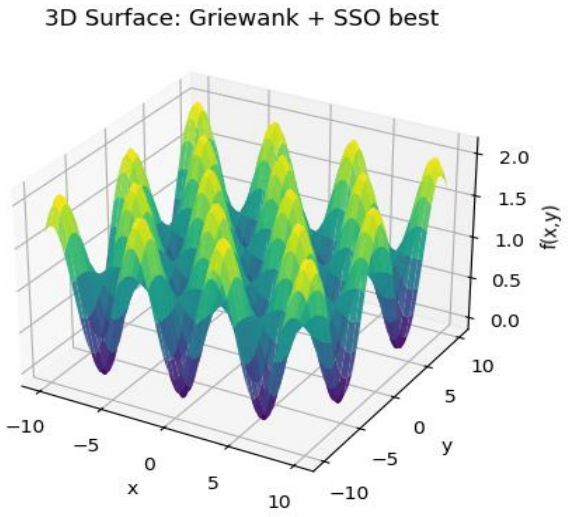

Figure-23-3D Surface: Griewank+SSO Best

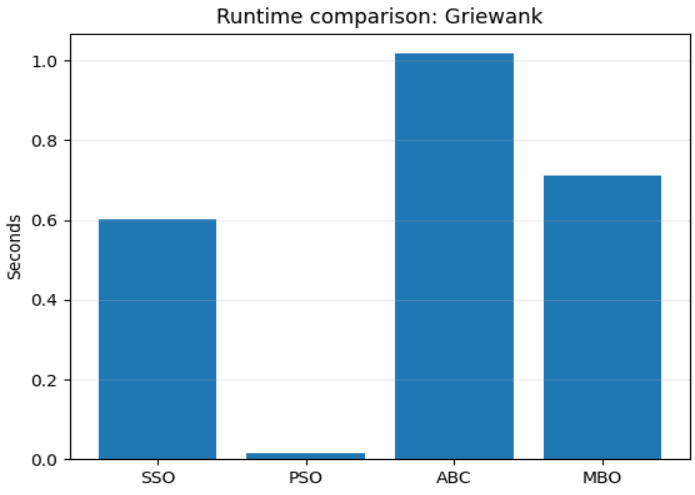

Figure-24-Runtime Comparison :Griewank

6. PSEUDOCODE AND FLOWCHART

6.1 SSO pseudocode

```
Input: f, bounds [l,u], population N, iterations T
Initialize x_i ~ Uniform(l,u), set p_i = x_i, g = argmin f(p_i)

for t = 1..T:
    r = 0.5*(1 + cos(2*pi*t/T))              # iambic rhythm
    for each i:
        pick a,b distinct from i
        s = x_i + alpha(t)*(p_i - x_i) + eta(t)*Normal(0,I)
        c = x_i + beta(t)*(g - x_i) + gamma(t)*(x_a - x_b)
        y = r*s + (1-r)*c

        with prob p_twist(t): y = x_i + delta(t)*Uniform([-1,1]^d)

        y = clip(y, l, u)

        if f(y) < f(x_i): x_i = y
        if f(x_i) < f(p_i): p_i = x_i
    g = best p_i
return g, f(g)
```

6.2 Simple flowchart (SSO)

```
Start
  |
Initialize population -> set personal bests -> set global best
  |
Loop t=1..T
  |
Compute rhythm r(t)
  |
For each agent i:
   |--> Soliloquy move s_i
   |--> Chorus move c_i
   |--> Blend y_i = r*s_i + (1-r)*c_i
   |--> (optional) Tragic twist jump
   |--> Clip to bounds
   |--> Accept-if-better + update pbest
  |
Update gbest
  |
End -> report gbest, runtime, plots
```

PS: The Python code is too long to include in the article to save space, but it can be sent upon request.

7. DISCUSSION

SSO's performance comes from its structured schedule: early differential-style exploration, mid-stage conflict push–pull between global destiny and personal memory, late neighborhood intensification, and final "couplet contraction." This is consistent with general metaheuristic wisdom: exploration early, exploitation late, diversity preserved via restart [6]–[8]. The oscillatory rhythm term provides mild periodicity that can reduce premature convergence in some landscapes (by preventing overly monotone step decay) [6], [7].

8. CONCLUSION

This paper demonstrated a systematic method for converting poetry or the personal traits of a famous figure into a mathematically defined metaheuristic optimizer. By treating poetic structure as a schedule for exploration and exploitation operators, we designed Shakespearean Soliloquy Optimization (SSO) and provided its full equations, pseudocode, and flowchart. On the convex test function $z = (x-5)^2 + (y-4)^2$, SSO reliably converges to the true optimum near (5,4) with a clear trajectory behavior. On five standard benchmark functions, SSO is competitive with classical optimizers (PSO, ABC) and a simplified MBO-like approach when run under identical evaluation budgets. The provided Python implementation (NumPy + Matplotlib only) produces multiple colorful plots including 3D colored surfaces, contour trajectories, convergence curves, and boxplots, enabling transparent visual comparison.

Future work may include: (i) adaptive phase boundaries based on online diversity measures, (ii) constraint handling via penalty–repair hybrids, (iii) multi-objective extensions using Pareto archives, and (iv) rigorous statistical testing (Wilcoxon/Friedman) across a larger benchmark set [2]–[4], [8].

References

[1] C. H. Wang, "Rethinking metaheuristics: Unveiling the myth of 'novelty in Metaheuristic Algorithms," *Mathematics* (MDPI), *13*(13), 2158, 2025.
[2] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*, 2nd ed., Luniver Press, 2010.
[3] J. M. Dieterich and B. Hartke, "Empirical review of standard benchmark functions using evolutionary global optimization," ,arXiv:1207-4318v1,18 jul 2012.
[4] M.Jamil and X.-S. Yang, "A literature survey of benchmark functions for global optimisation," *International Journal of Mathematical Modelling and Numerical Optimisation.* 4 (2), pp. 150-194, 2013.
[5] D. H. Wolpert and W. G. Macready, "No Free Lunch Theorems for Optimization," *IEEE Transactions on Evolutionary Computation*, Vol 1(1),1997.
[6] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, Perth, Australia, 1995, pp. 1942–1948.
[7] D. Karaboga, *An Idea Based on Honey Bee Swarm for Numerical Optimization*, Technical Report TR06, Erciyes University, 2005.
[8] D. Karaboga and B. Akay, "A Comparative Study of Artificial Bee Colony Algorithm," *Applied Mathematics and Computation*, 214, 108-132,2009.
[9] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial Bee Colony (ABC) algorithm," *Journal of Global Optimization*, 39, 459-471,2007.
[10] E. Duman, M. Uysal, and A. F. Alkaya, "Migrating Birds Optimization: A new metaheuristic approach and its performance on the quadratic assignment problem," Information Sciences,(217), 65-77, 2012.
[11] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, pp. 175–184, 1960.
[12] L.Ngartera," A Comparative Study of Optimization Techniques on the Rosenbrock Function", Open Journal of Optimization Vol.13(No.3),51-63,2024.

[13] L.A.Rastrigin," Random Search in Problems of Optimization, Identification and Training of Control Systems ,"*journal of Sybernetics,* Vol.3(3), 1973.
[14] W.Cai,Li Yang et al." Solution of ackley function based on particle swarm optimization algorithm*," IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*,DOI: 10.1109/AEECA49918.2020,25-27 Aug. 2020.
[15] M.Locatelli," A Note on the Griewank Test Function,"*Journal of Global Optimization,* Vol 25, pages 169–174, 2003

APPENDIX:OUTPUT OF THE PYTHON CODE

C:\Users\Lenovo\PycharmProjects\PythonProject20\.venv\Scripts\python.exe
C:\Users\Lenovo\PycharmProjects\PythonProject20\.venv\Scripts\activate_this.py

=== Quadratic (x-5)^2+(y-4)^2 ===

Method | best f(x*) | x* | runtime (s)

SSO | 8.251802e-12 | [5.00000174 4.00000229] | 0.4229

PSO | 3.914722e-26 | [5. 4.] | 0.0099

ABC | 0.000000e+00 | [5. 4.] | 0.6355

MBO | 5.711068e-12 | [4.99999805 4.00000138] | 0.3991

=== Sphere ===

Method | best f(x*) | x* | runtime (s)

SSO | 6.833223e-12 | [-2.52834431e-06 6.63850818e-07] | 0.5692

PSO | 1.037195e-30 | [-8.84771049e-18 1.01838929e-15] | 0.0149

ABC | 6.357529e-89 | [-6.12430685e-45 5.10569792e-45] | 0.9114

MBO | 1.215919e-11 | [-2.40230096e-06 2.52747756e-06] | 0.5859

=== Rosenbrock ===

Method | best f(x*) | x* | runtime (s)

SSO | 1.035747e-08 | [1.00008662 1.00016791] | 0.6060

PSO | 7.750040e-17 | [0.99999999 0.99999998] | 0.0126

ABC | 8.301656e-07 | [0.99928935 0.99863623] | 0.9291

MBO | 8.103924e-10 | [0.99997568 0.99994989] | 0.5682

=== Rastrigin ===

Method | best f(x*) | x* | runtime (s)

SSO | 1.404066e-07 | [9.91387377e-06 2.46867920e-05] | 0.6912

PSO | 0.000000e+00 | [1.42697084e-10 1.41695045e-09] | 0.0174

ABC | 0.000000e+00 | [4.42128561e-10 -5.59238047e-10] | 0.9942

MBO | 1.129422e-08 | [-6.76678874e-06 3.33756597e-06] | 0.5200

=== Ackley ===

Method | best f(x*) | x* | runtime (s)

SSO | 6.343472e-06 | [1.89985035e-06 1.19176769e-06] | 0.6509

PSO | 3.996803e-15 | [6.64546333e-16 -3.24445307e-16] | 0.0147

ABC | 4.440892e-16 | [-2.44840810e-16 -1.15108788e-16] | 1.0946

MBO | 2.104719e-05 | [-5.31845842e-06 5.20377646e-06] | 0.8249

=== Griewank ===

Method | best f(x*) | x* | runtime (s)

SSO | 1.669777e-07 | [-0.0001499 -0.00078888] | 0.6010

PSO | 0.000000e+00 | [-1.04818159e-09 4.44890646e-09] | 0.0151

ABC | 4.593329e-05 | [-0.00788932 -0.00768972] | 1.0176

M

MBO | 4.349088e-11 | [4.18581819e-06 -1.17798539e-05] | 0.7122