SocioCyclic Optimizer: a Novel Swarm-Based Metaheuristic Algorithm

MOHAMMED H.S. HELAL
Department of Computer Science
Birzeit University
Ramallah,
PALESTINE
ORCID: 0000-0001-6898-9650

Abstract: This paper presents a novel swarm evolutionary metaheuristic optimization algorithm inspired by the cyclic nature of human civilization in its rise from nomadic life to the point where it peaks, stagnates, and then declines back to nomadic life. Nomads do the exploration, while civics do the exploitation. The proposed algorithm also mimics basic attributes and behaviors of human civilizations from hostility that pushes civilizations away from each other to cooperation and trade which helps to explore areas in between and exchange knowledge of best traits (products). Experimental results show highly competitive results when benchmarked with common swarm-based evolutionary algorithms such as Genetic Algorithm, Particle Swarm Optimization, Wolf Pack Algorithm, and Artificial Bee Colony, using common test functions like Rastrigin, Schwefel, Rosenbrock, Griewank, and Ackley.

Key-Words: Metaheuristic Optimization, Swarm Intelligence, Swarm-based Optimization, Evolutionary Algorithms, Evolutionary Optimization, Numerical Function Optimization.

Received: October 25, 2025. Revised: December 16, 2025. Accepted: January 11, 2026. Published: May 5, 2026.

1 Introduction

Mathematical function optimization has been used to optimize a wide variety of industrial and day-to-day applications, for instance, traffic light synchronization, [1], college exam scheduling, [2], software test suite generation, [3], VLSI layout, [4], robotics, [5], power grid optimization, [6] and even the make up of alloys and semiconductor materials, [7]. Basically, if you can formulate a problem into a mathematical or logical function with a set of input parameters, then function optimization methods can be used to find the set of inputs that generate the optimal function value (i.e. the output). The input parameters represent the different settings and configurations that need to be optimized, and the function represents the quality or expected outcome of the input parameter. For some problems, the function can be rather complex and the set of inputs can have a large number of parameters with a large range of possible values for each parameter. For such problems, it might be impossible to find the optimal solution in a reasonable time frame. Such applications are often optimized using Metaheuristic Optimization algorithms.

The literature has explored a huge variety of metaheuristic optimization algorithms with various categories and classifications. The most common category is the Combinatorial approach, which itself is divided into numerous subcategories with one shared characteristic; that is, following the pattern of trying different combinations of input proposals and modifying the proposed sets according to particular strategies and mechanisms. Some propose a single proposal at a time and some keep a predetermined number of proposed solutions at a time, the latter can be either Evolutionary or Swarm-based. Evolutionary Algorithms (EA) propose a set of solutions and then evolve them over time until they produce desirable results; the most common example is the Genetic Algorithm (GA), [8]. GA was developed in the 1970s and is one of the most studied approaches. It is inspired by natural evolution where an initial 'population' of solution is proposed, each member of the population (i.e. a proposed solution for the problem) is called a Chromosome, then the natural evolution steps of evaluation, selection, crossover and mutation are simulated for many generations until the termination condition is met, over generations, quality of the population keeps improving. Swarm-based Optimization (SO), on the other hand, is inspired by the collective behavior of groups in nature, for example bird flocks, wolf packs, ant and bee colonies.

SO along with GA have gained ground from day one and they have quickly become the most common approaches to be explored and employed. Researchers have presented plenty of SO algorithms inspired by many different natural swarms; for instance, Artificial Bee Colony (ABC), [9], Particle Swarm Optimization (PSO), [10] and Wolf Pack Algorithm (WPA), [11]. ABC is inspired by bee kingdoms and their designation of tasks for onlooker bees and worker bees. PSO is inspired by bird flocks and fish schools where the leader of the group influences the direction of their motion, [10]. WPA is inspired by wolf packs when they attempt to take on a

target (i.e. point of interest) from multiple directions, moreover, the division of the pack into alpha, beta, delta and omega, each with different responsibility, [11]. Plenty of modern variants of WPA has been developed in recent years, in this paper we will feature OGL-WPA introduced by [12], the algorithm uses Opposition-based learning to preserve diversity of the pack and therefore provide a wide coverage for search space.

For a search algorithm to be successful, it has to in a way or another execute the two following tasks: exploration and exploitation. Exploration is when individuals travel quickly and cover a large search space, which allows the algorithm to escape the local optima traps, while exploitation helps further tuning and improvement around the best scouted point or points of interest. For ABC, the colony is divided into scout bees and worker bees, scout bees do the exploration and worker bees do the exploitation. For WPA, the whole pack does both tasks on different intervals, the balance between the two phases is dynamically managed rather than dedicating specific individuals for each task. OGL-WPA improves WPA's exploration by preserving the diversity of the pack. PSO does not have a particular mechanism to differentiate between exploration and exploitation, the motion of each individual is influenced by a combination of its own memory (momentum) and by attraction to the best known position by the group (the leader).

This paper introduces the SocioCyclic Optimizer (SCO), a novel swarm-based method inspired by the cyclic dynamics of human civilizations. SCO balances exploration and exploitation by alternating the population between nomadic and civic modes. Nomads explore rapidly and broadly until they locate promising resources; they then form new civilizations. Citizens cluster around capitals and exploit local resources until stagnation (i.e., no further improvement) triggers collapse back into nomadic life.

2 The SocioCyclic Optimizer

In his Muqaddimah, [13], Ibn Khaldoun (1331-1406 CE) has done extensive studies on the cyclic nature of civilizations between rise from nomadic lifestyle and collapse, often back to nomadic lifestyle and sometimes even to extinction. He observed the phenomena and studied plenty of examples through history known to his time and he tried to explain the socio-economic factor that drives the cycle, [14], what motivates the establishment of a civilization and what causes it to decline. Ibn Khaldoun has summarized a civilization's sign of weakness using a famous parody: at the early stage of a civilization, it achieves a lot with little tax (i.e. resources), while at its late stage it achieves little with lots of tax, [13]. Inspired by his work, this research paper proposes a mathematical function optimization algorithm that imitates the cyclic nature of human civilization between nomadic life, followed by establishment of civilization, and finally, collapse often back to nomadic life after stagnation. The algorithm also simulates the complex relationship between civilizations, namely hostility and trade. Hostility drives civilizations away from each other, which helps expand the search space. While trade is simulated by trading the best traits each civilization has to offer, where traits represent the dimensions of the search space, the equivalent of 'genes' in GA. the next sections will provide an indepth overview on the proposed algorithm.

2.1 Problem Setting and Notation

We consider bound-constrained black-box minimization shown in Equation 1:

$$\min_{x\in[\ell,u]^d} f(x), \tag{1}$$

where $x \in \mathbb{R}^d$ and bounds are given per dimension by ℓ_j, u_j.

The population size is N, the number of generations is G, the stagnation window is W, and the rebirth intensity is $R \in [0, 1]$.

2.2 SCO Overview

SCO partitions the population into two roles: *nomads* (global explorers) and *citizens* (local exploiters grouped into civilizations). Each civilization maintains a *capital* (its best member), collaborates locally to exploit the surrounding region, and adapts a city-level step size. When a civilization stagnates, it collapses and its members revert to nomads. High-quality nomads can found new civilizations. A periodic guarded *hub exchange* allows limited information transfer between cities. The best (champion) civilization is protected from collapse to prevent catastrophic forgetting.

2.3 Data Structures

Agent. Each agent stores its position $x \in \mathbb{R}^d$, fitness $f(x)$, role (NOMAD or CITIZEN), and a city identifier.

Civilization. A civilization C stores its member indices, capital c, best fitness f_C, a diagonal scale per-dimension s (estimated by MAD among members), a radius ρ (mean distance of members to the capital), stagnation counters, and a city step size α_C.

Ruin. A ruin is defined by a center, radius, and expiry generation, and prevents immediate re-founding at the same location after collapse.

2.4 Algorithm Steps

The following subsections will cover the algorithm steps.

2.4.1 Initialization

All N agents are sampled uniformly in $[\ell, u]^d$ and initialized as nomads. Fitness values are evaluated and the global best g is recorded. The civilization and ruin sets are initially empty.

2.4.2 Nomad Phase (Exploration)

Each nomad performs a global exploratory[] step that combines attraction toward the global best, interaction with a random peer, and Gaussian jitter. The move is accepted if fitness improves. The nomadic motion is explained in details in the next paragraph.

Nomad motion Nomads perform global exploration using a PSO-like, memory-less step that mixes attraction to the global best $g(t)$ and a randomly sampled peer $x_r(t)$, plus jitter. Nomad exploration is illustrated by Equation 2:

$$\begin{aligned}\tilde{x}_i(t) \;=\; \mathcal{R}\Big(x_i(t) &+ c_1\, r_1(t) \\ &\odot \big(g(t) - x_i(t)\big) + c_2\, r_2(t) \\ &\odot \big(x_r(t) - x_i(t)\big) + \sigma_N(t)\, \varepsilon_i(t)\Big).\end{aligned} \tag{2}$$

where $r_1(t), r_2(t) \sim U([0,1]^d)$ element-wise, $\varepsilon_i(t) \sim \mathcal{N}(0, I_d)$, $c_1, c_2 > 0$ are fixed gains, and $\sigma_N(t)$ is a nomad jitter scale.

The nomad update in greedy manner as per Equation 3, the nomad makes a choice between value resulted from Equation 2 or not doing any change if fitness not improved:

$$x_i(t{+}1) \;=\; \begin{cases} \tilde{x}_i(t), & \text{if } f(\tilde{x}_i(t)) < f(x_i(t)), \\ x_i(t), & \text{otherwise.} \end{cases} \tag{3}$$

City Founding A nomad whose fitness lies within a top quantile and which is sufficiently far from existing capitals and ruins may found a new civilization.

Equation 4 shows how the threshold value for extablishing a new civilization is calculated, and Equation 5 shows how the threshold is used to determine if a new civilization is to be established. For Equation 4, let i be a nomad at generation t with position $x_i(t)$ and fitness $f_i(t)$. Define a founding fitness threshold as the $q_{\text{found}}(t)$-rank-based fitness cutoff:

$$\tau(t) = Q_{q_{\text{found}}(t)}\big(\{f_j(t)\}_{j=1}^{N}\big)\,. \tag{4}$$

Nomad i may found a new civilization if

$$\begin{aligned} f_i(t) \le \tau(t) \;\;\wedge\;\; \min_{C} \frac{\|x_i(t) - c_C(t)\|_2}{\max(\rho_C(t), \varepsilon)} > \kappa \\ \wedge\;\; \min_{r} \|x_i(t) - z_r(t)\|_2 > \rho_r, \end{aligned} \tag{5}$$

where c_C and ρ_C are existing city capitals and radii, and (z_r, ρ_r) are active ruins.

Let $r_{\text{init}}(t)$ be the median distance of the top q_{found} individuals to their centroid. Upon founding, a nomad i initializes a new civilization by recruiting the $K-1$ nearest nomads in the search space, such that the initial population of the civilization comprises the founder and its closest neighbors. The new civilization consists of the founder plus its closest neighbors in the search space. Equation 6 shows the new city C_{new} being initialized by recruiting the $K-1$ nearest nomads within $r_{\text{init}}(t)$:

$$\begin{aligned}\text{memb}(C_{\text{new}}) = \{i\} \cup \\ \operatorname*{arg\,min}_{j_1,\ldots,j_{K-1}} \sum_{k=1}^{K-1} \|x_{j_k}(t) - x_i(t)\|_2\end{aligned} \tag{6}$$

The capital is set to the best recruited member as shown in Equation 7:

$$c_{C_{\text{new}}}(t) = \arg \min_{j \in \text{members}(C_{\text{new}})} f(x_j(t)). \tag{7}$$

2.4.3 City Phase (Exploitation)

Citizen Updates Each citizen proposes two full-dimensional candidate moves constructed from: (i) attraction toward the capital, and (ii) Gaussian noise scaled by $\alpha_C s$. With probability p_{DE}, a DE/best/1 proposal centered at the capital is also attempted. The best candidate is greedily accepted.

Motion of Individuals Motion within cities have two different behaviors, the first is city polishing and the second is citizen motion. Let $x_i(t) \in \mathbb{R}^d$ be the position of individual i at generation t, and let $f(\cdot)$ be minimized. Bounds are handled by a component-wise reflection operator $\mathcal{R}(\cdot)$ that maps any vector back into $[\ell, u]^d$.

Citizen motion Each civilization C maintains a capital (best solution) $c_C(t) \in \mathbb{R}^d$ and a per-dimension spread vector $s_C(t) \in \mathbb{R}^d_+$ (estimated, e.g., by MAD across citizens), as well as a city step scale $\alpha_C(t) > 0$.

For a citizen $i \in C$, define the attraction (pull) vector by Equation 8:

$$v_i(t) \;=\; c_C(t) - x_i(t). \tag{8}$$

SCO generates two candidate proposals and greedily accepts the best one. For proposal $k \in \{1, 2\}$, Equation 9 shows citizen update choising between bot k values:

$$\begin{aligned}\tilde{x}_i^{(k)}(t) = \mathcal{R}\Big(x_i(t) + \beta_k\, v_i(t) \\ + \alpha_C(t)\,\big(s_C(t) \odot \varepsilon_i^{(k)}(t)\big)\Big).\end{aligned} \tag{9}$$

where $\odot$ denotes element-wise multiplication, $\varepsilon_i^{(k)}(t) \sim \mathcal{N}(0, I_d)$, and typical choices are $\beta_1 = 1$ and $\beta_2 = \frac{1}{2}$ (a stronger and a milder pull). Equation 10 shows how the noise vector is orthogonalized to the pull direction to avoid redundant movement:

$$\varepsilon \;\leftarrow\; \varepsilon - \frac{\varepsilon^\top v}{\|v\|^2 + \delta}\, v, \tag{10}$$

with a small $\delta > 0$ for numerical stability.

With probability p_{DE}, an additional DE/best/1-style trial is attempted using two distinct citizens $r_1, r_2 \in C$ as per Equation 11:

$$\tilde{x}_i^{(\text{DE})}(t) = \mathcal{R}\begin{pmatrix} c_C(t) \\ + F\big(x_{r_1}(t) - x_{r_2}(t)\big)\end{pmatrix}. \tag{11}$$

where $F > 0$ is a fixed scaling factor.

Finally, Equation 12 and Equation 13 show the citizen updates by greedy selection between $\tilde{x}_i^{(k)}(t)$ generated by Equation 9, $\tilde{x}_i^{(\text{DE})}(t)$ generated by Equation 11, and making no change:

$$\mathcal{X}_i(t) = \big\{x_i(t),\, \tilde{x}_i^{(1)}(t),\, \tilde{x}_i^{(2)}(t),\, \tilde{x}_i^{(\text{DE})}(t)\big\}, \tag{12}$$

$$x_i(t{+}1) = \arg\min_{x \in \mathcal{X}_i(t)} f(x). \tag{13}$$

(where the DE candidate is included only when it is sampled).

Capital polishing In addition to individual citizen updates, each city performs a low-cost coordinate-block polish on its capital. Let $J(t) \subseteq \{1, \ldots, d\}$ be a small random subset of M dimensions. Equation 14 defines a step per selected coordinate

$$\begin{aligned}h_j(t) \;=\; \max(\eta\,(u_j - \ell_j),\, \kappa\,\alpha_C(t)\,s_{C,j}(t))\,, \\ j \in J(t),\end{aligned} \tag{14}$$

with constants $\eta \in (0, 1)$ and $\kappa > 0$. Construct two trials as illustrated by Equation 15:

$$\begin{aligned}c_C^{(+)}(t) &= \mathcal{R}\left(c_C(t) + \sum_{j \in J(t)} h_j(t)\, e_j\right) \\ c_C^{(-)}(t) &= \mathcal{R}\left(c_C(t) - \sum_{j \in J(t)} h_j(t)\, e_j\right)\end{aligned} \tag{15}$$

where e_j is the jth basis vector, and update is done greedily by Equation 16 and Equation 17:

$$\mathcal{Z}_C(t) = \big\{c_C(t),\, c_C^{(+)}(t),\, c_C^{(-)}(t)\big\}, \tag{16}$$

$$c_C(t{+}1) = \arg\min_{z \in \mathcal{Z}_C(t)} f(z). \tag{17}$$

Stagnation and Collapse If a city fails to improve for W_{eff} generations,
where $W_{\text{eff}} = \text{round}(W(1 - 0.7R))$, it is marked for collapse. In order to preserve global progress, champion city is exempt from collapse.

Collapse and dispersal When city C collapses, each former citizen $i \in C$ is dispersed around the last capital using a Gaussian kernel scaled by city spread as illustrated by Equation 18:

$$\begin{aligned}x_i(t^+) \;=\; \mathcal{R}(c_C(t) + \Sigma_C(t)\,\xi_i) \\ \xi_i \sim \mathcal{N}(0, I_d),\end{aligned} \tag{18}$$

where $\Sigma_C(t) = \text{diag}(\sigma_{C,1}(t), \ldots, \sigma_{C,d}(t))$ where $\sigma_{C,j}(t)$ is calculated by Equation 19:

$$\sigma_{C,j}(t) \;=\; \max(1.5\, s_{C,j}(t),\; 0.10\,(u_j - \ell_j))\,. \tag{19}$$

This dispersal restores diversity and supplies nomads to explore new basins.

step-size adaptation At the city level, the step scale α_C is adapted using a 1/5 success heuristic. As per Equation 20, let $\text{SR}_C(t)$ be the fraction of citizens in C that improved at generation t. Then

$$\alpha_C(t{+}1) = \text{clip}\begin{pmatrix}\alpha_C(t)\,\gamma^{\text{sign}(\text{SR}_C(t) - 0.2)}, \\ \alpha_{\min}, \alpha_{\max}\end{pmatrix} \tag{20}$$

with $\gamma > 1$ and clipping bounds $(\alpha_{\min}, \alpha_{\max})$.

2.4.4 Hub Gene Exchange

Every fixed number of generations, each city estimates a signature gene. The top two cities broadcast their signature gene values. A receiving city adopts an imported gene only if it strictly improves the capital.

2.5 SCO Exploration and Exploitation

In order for an optimization algorithm to heavily invest the CPU time in improving its best known positions, and at the same time to escape the trap of local optima, the algorithm has to balance between exploration and exploitation. As mentioned in introduction section, swarm-based algorithms often dedicate particles for each task which results in adding a setting parameter that requires tuning in order to guarentee efficiency and efficacy. SCO on the other hand proposes two different opproaches in order to balance between exploration and exploitation, the first approach is allowing the individuals to go throw different phases of exploration as nomads or exploitation as citizens without the need to predetermined parameter setting. The other approach is the balance between inter-civilizations relationships of trade and hostility. Trade is done "gene" exchange, which results in driving cities to the same position helping further inprovement, and then hostility pushes them away from each helping to expand search space. The combination of the two approaches of cyclic dynamic and the inter-civilisational relationships on the algorithm's capity to navigate hills and vallyes of the search space is to be put under test by benchmarking with popular optimization algorithms and results will be presented in the experimental results section.

2.6 Algorithm Complexity

Per generation, SCO evaluates $\mathcal{O}(Nd)$ candidate solutions.

2.7 Algorithm Summary

The psudocode shown in Table 1 summarizes the basic operations and steps of SCO algorithm. The psudocode shows the main loop iterating over each individual. Basic SCO algorithm steps are done using conditional statements within the main loop.

The next section will cover experimental design, setup, benchmarking and results.

3 Experimental Setup and Results

The proposed algorithm has been implemented using Java and it has been tested on common test functions. Results has been benchmarked with GA, PSO, ABC and WPA in terms of convergence rate as well as quality of final solutions.

3.1 The Test Functions

Table 1 (Appendix) shows details on the test functions used for the comparison, the selected functions are commonly used in the literature and they provide a diversity of obstacles that challenge optimization functions.

Table 1: SocioCyclic Optimizer (SCO) — summarizing pseudocode. *Source: prepared by author.*

Algorithm (SCO)
Initialize and evaluate population $\{x_i(0)\}_{i=1}^{N}$.
Set global best.
For $t = 1$ **to** G **do**
  For all cities $C \in \mathcal{C}(t)$ **do**
    Perform citizens motion in C.
    Perform city polishing for C.
    If C is due to collapse **and** is not global best **then**
      Convert C to a ruin.
      Convert citizens of C to nomads.
    End if
  End for
  For all nomads i with $S_i(t) = \text{NOMAD}$ **do**
    Perform nomadic motion for i.
    If good resources found **and** no ruins are near **then**
      Create a new city and assign i as a founder (citizen).
    End if
  End for
  For all ruins $r \in \mathcal{R}(t)$ **do**
    If r is expired **then** Delete r. **End if**
  End for
End for

Rastrigin, Griewank and Ackley have plenty local optima, which can trap the algorithms and hinder them form finding the optimak solution. While Schwefel 2.21 and Rosenbrock test the algorithm's ability to find the optimal solution in the shortest time or number of iterations.

3.2 Parameter Tuning

The settings parameters for GA are taken from, [15], ABC is from, [16], while PSO, WPA and OGL-WPA are taken from, [12], according to the sources, the same parameters have been used for all five test function. Table 2 (Appendix) shows the parameter setting for GA, PSO, ABC, WPA and OGL-WPA.

For SCO, two different sets of parameters have been used, a set for f_1, f_2 and f_4, and another set for f_3 and f_5. Experimental work done to chose these parameters is shown in a seperate subsection. Table 2 shows parameters for SCO. For the parameter N, the choice has been made similarly to the other algorithms for sake of fareness of comparison.

3.3 Experimental Results

This section presents the conducted experimental results and it is divided into three subsections: firstly we explore parameter tuning and effect of parameters on the convergence rate. Then we present

Table 2: Setting parameters for SCO. *Source: Created by author.*

Parameter	f_1, f_2 **and** f_4	f_3 **and** f_5
N	100	100
W	50	500
R	0.2	0.8

a benchmarking experiment to compare the proposed algorithm with common swarm-based optimization algorithms. Finally, the SCO performance is illustrated in term of CPU time.

3.3.1 Convergence Rate and Parameter Selection

In this section, the effect of parameter tuning on the convergence rate is ellustrated and the choice for the parameters is justified. As mentioned earlier, the parameters used to optimize test functions may vary and the choice should be made after experimental analysis, for sake of conciseness, and similarly to many related papers in the literature, we will only show the convergence rate analysis on f_2, that is Rastrigin function.

First we begin with the parameter N, which represents the total number of population. The expected outcome from increasing the population is direct improvement on the convergence rate, and that is because the more individuals moving in the search space will lead to more coverage per iteration. However, this will be true for a certain limit where the effect of the other parameters could play a bigger role in effecting the convergence. In order to domenstrate the effect of N on the convergence rate, SCO is executed with different suggested N values (50, 100, 200 and 300) with 30 runs for each, then the average of the best solution known at an interval of 100 iterations is recorded and plotted in Figure 1. The figure shows that SCO performance peeks at $N = 200$. Its iportant to mention that the values of the other parameters W and N are fixed at the best values known, which are 50 and 0.2 respectively.

Secondly, we study the effect of the parameter W on convergence rate. W decides the stagnation window, the smaller it is, the more frequent is the civilizational collapse, and as a result, the better exploration at the expense of exploitation. Thats why W has a strong influence on the algorithm's performance, and should optimized for each test function or practicale optimization problem. To domenstrate the effect of W on convergence rate, SCO is executed with different suggested W values (10, 30, 50 and 70) with 30 runs for each, then the average of the best solution known at an interval of 100 iterations is recorded and plotted in Figure 2.

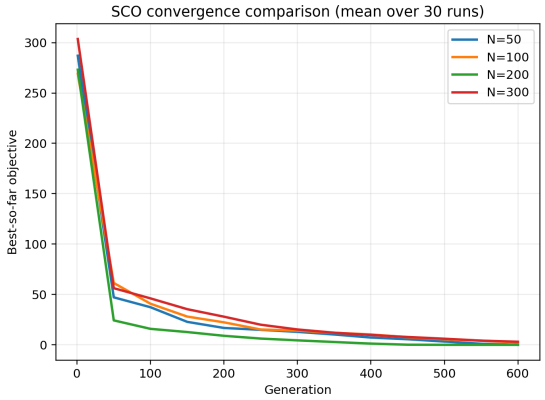

Fig. 1: Effect of varying N on SCO convergence. *Source: Author's experiments.*

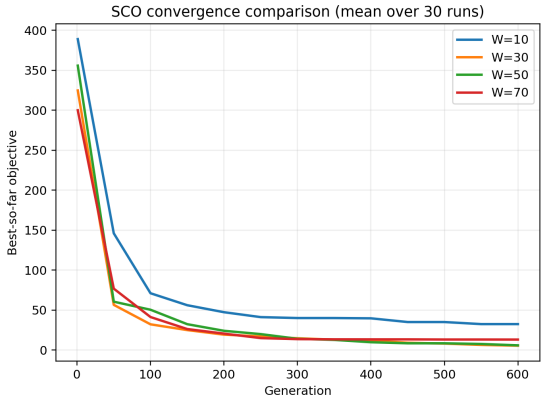

Fig. 2: Effect of varying W on SCO convergence. *Source: Author's experiments.*

The values for N and R are fixed at 100 and 0.2 respectively. The figure shows that SCO performs best at $W = 50$ and $W = 30$.

And now for the R parameter, it controls how aggressively do civilizations collapse, the higher the value, the stronger the dispersal of citizens when their civilization is collapsed. To illustrate the effect of R on convergence and to decide what value is to be adopted for f_1, the algorithm is executed with different suggested R values (0.2, 0.4, 0.6 and 0.8) with 30 runs for each, then the average of the best solution known at an interval of 100 iterations is recorded and plotted in Figure 3. The values for N and W are fixed at 100 and 50 respectively.

Figure 3 shows that SCO performed best at $R = 0.2$ for f_1. Figure 1, Figure 2 and Figure 3 show comparison on convergence rate under different parameter settings, however, they dont show the exact values of the found solutions since its difficult show such precise values. The next section shows the exact values of the final outputs with comparison with dominant algorithms.

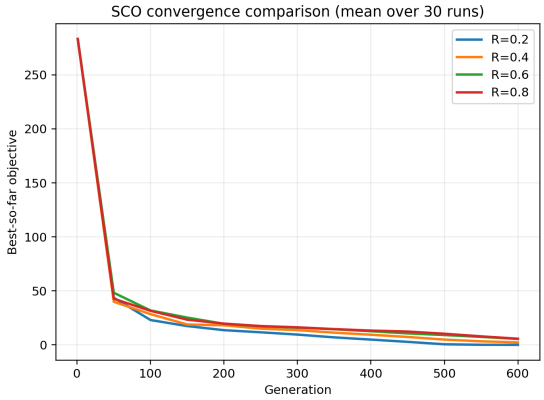

Fig. 3: Effect of varying R on SCO convergence. *Source: Author's experiments.*

3.3.2 Benchmarking

The SCO has been compared with GA, PSO, GWO, ABC and OGL-WPA, each running 30 times using different seeds, and the mean and standard deviation of the best acquired results have been recorded. Each execution is done for 1000 iterations, with the parameter settings mentioned in Table 2 (Appendix). Results for GA have been taken from, [15], ABC have been taken from, [16], while PSO, WPA and OGL-WPA have been taken from, [12].

Table 3 (Appendix) presents the comparison between the aforementioned algorithms optimizing the five test functions in different dimensionality (5, 10 and 30 dimensions).

According to results shown in Table 3 (Appendix), SCO and ABC managed to achieve the best results for the most part, and OGL-WPA did perform better in one of the test function with 30 dimensions. Best mean values for each implementation has been highlighted with bold font style.

For f_1, SCO has produced the best results for dimensionalities 5 and 10, while ABC did better in 30 dimensions. As for f_2, SCO has done the best for all tested dimensionalities. That is an expected result as the algorithm has been optimized to avoid falling into local optima trap, a feature that is prevalent in Rastrigin function (f_2). SCO has also achieved best results for lower dimensionality of f_3 but fell in second place behind ABC for higher dimensionality. This might be a predictable result as well, since ABC does better focus on exploitation and Rosenbrock function (f_3) has no local optima traps which makes PSO's feature of hostility between civilizations less beneficent. For f_4, once again, SCO has scored best results for lower dimensionality, and then ABC did better on dimensionality 30. Finally, for f_5, OGL-WPA has done best for high dimensionality, with SCO falling in second place. ABC did the best on lower dimensionality. Schwefel 2.21 (f_5), shares the same characteristic with f_3, its an exhaustive function with slope sliding to the global optima, a search space with no local optima traps.

Table 3: SCO Execution time in ms across different dimensionalities. *Source: Author's experiments.*

	5d	**10d**	**30d**
f_1	199	280	664
f_2	165	236	492
f_3	150	211	432
f_4	157	229	521
f_5	119	191	453

3.3.3 Experiment environment and execution time

The experiments have been performed on a Macbook Pro machine with M4 Chip and 48 GB of RAM. In order to verify SCO's time complexity and domenstrate its efficiency, the average execution time in milliseconds for 30 different runs of each test function with the three different dimensionalities of 5, 10 and 30 dimensions has been captured and summarised in the Table 3.

4 Conclusion

This paper has presented the design and implementation of a novel swarm-based metaheuristic algorithm inspired by the cyclic nature of human civilizations, the cycle of rise from wondering nomads, to establishment of civilizations, and then stagnation and collapse back nomadic lifestyle. The algorithm also mimics the complex relationships between civilizations, of rivalry and cooperation. The proposed algorithm has been compared with common swarm-based optimization algorithms running on common test functions, and benchmarking results demonstrates the efficacy of the the proposed algorithm as it has outperformed the other algorithms, specially for test functions with rough terrain, with multiple local optima traps.

References

[1] E. Sofronova, "Evolutionary computations for traffic signals optimization," *Procedia Computer Science*, vol. 186, no. 1, pp. 802–811, 2021, 14th International Symposium "Intelligent Systems, ISSN: 1877-0509. DOI: 10.1016/j.procs.2021.04.202. [Online]. Available: https://doi.org/10.1016/j.procs.2021.04.202.

[2] M. Freire, E. Soboredo, and M. Pedemonte, "Designing higher education exam schedules with multiobjective evolutionary algorithms," in *2025 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, Mexico City, Mexico, 2025, pp. 1–6. DOI: 10.1109/LA-CCI66231.2025.11270432. [Online]. Available: https://doi.org/10.1109/LA-CCI66231.2025.11270432.

[3] J. Campos, Y. Ge, N. Albunian, G. Fraser, M. Eler, and A. Arcuri, "An empirical evaluation of evolutionary algorithms for unit test suite generation," *Information and Software Technology*, vol. 104, no. C, pp. 207–235, Dec. 2018, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2018.08.010. [Online]. Available: https://doi.org/10.1016/j.infsof.2018.08.010.

[4] J. Cohoon, J. Kairo, and J. Lienig, "Evolutionary algorithms for the physical design of vlsi circuits," in *Advances in Evolutionary Computing: Theory and Applications*, A. Ghosh and S. Tsutsui, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 683–711, ISBN: 978-3-642-18965-4. DOI: 10.1007/978-3-642-18965-4_27. [Online]. Available: https://doi.org/10.1007/978-3-642-18965-4_27.

[5] A. Al Dahoud, A. Al Dahoud, M. Fezari, and H. Mimi, "Implementing swarm intelligence for mobile robots: Exploration of core algorithms, multi-domain applications, and future challenges," *WSEAS Transactions on Circuits and Systems*, vol. 24, pp. 310–316, 2025. DOI: 10.37394/23201.2025.24.32. [Online]. Available: https://doi.org/10.37394/23201.2025.24.32.

[6] F. Zaro and S. Altakrouri, "Optimal sizing and placement of distributed generators using multi-objective particle swarm optimization," *WSEAS Transactions on Power Systems*, vol. 20, pp. 401–411, 2025. DOI: 10.37394/232016.2025.20.32. [Online]. Available: https://doi.org/10.37394/232016.2025.20.32.

[7] C. E. Mohn, S. Stølen, and W. Kob, "Predicting the structure of alloys using genetic algorithms," *Materials and Manufacturing Processes*, vol. 26, no. 3, pp. 348–353, 2011, ISSN: 1532-2475. DOI: 10.1080/10426914.2011.552021.

[8] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley Professional, Jan. 1989, ISBN: 0201157675. Accessed: Mar. 6, 2026. [Online]. Available: https://books.google.com/books?vid=ISBN978-0-201-15767-3.

[9] D. Karaboğa and B. Baştürk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007. DOI: 10.1007/s10898-007-9149-x. [Online]. Available: https://doi.org/10.1007/s10898-007-9149-x.

[10] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007, ISSN: 1532-2475. DOI: 10.1007/s11721-007-0002-0. [Online]. Available: https://doi.org/10.1007/s11721-007-0002-0.

[11] H.-S. Wu and F.-M. Zhang, "Wolf pack algorithm for unconstrained global optimization," *Mathematical Problems in Engineering*, vol. 2014, no. 465082, pp. 1–17, 2014. DOI: 10.1155/2014/465082. [Online]. Available: https://doi.org/10.1155/2014/465082.

[12] X. Chen, F. Cheng, C. Liu, L. Cheng, and Y. Mao, "An improved wolf pack algorithm for optimization problems: Design and evaluation," *PLOS ONE*, vol. 16, no. 8, e0254239, 2021. DOI: 10.1371/journal.pone.0254239. [Online]. Available: https://doi.org/10.1371/journal.pone.0254239.

[13] I. Khaldun, *The Muqaddimah: An Introduction to History*, trans. by F. Rosenthal. Princeton, NJ: Princeton University Press, 1958, Translated from the Arabic; commonly cited edition (3 vols.).; ISBN corresponds to a later edition/reprint., ISBN: 0691097976. Accessed: Mar. 6, 2026. [Online]. Available: https://books.google.com/books/about/The_Muqaddimah.html?id=yyZtAAAAMAAJ.

[14] J. D. C. Boulakia, "Ibn khaldun: A fourteenth-century economist," *Journal of Political Economy*, vol. 79, no. 5, pp. 1105–1118, 1971. DOI: 10.1086/259818. [Online]. Available: https://doi.org/10.1086/259818.

[15] D. Srinivasan and T. H. Seow, "Particle swarm inspired evolutionary algorithm (ps-ea) for multiobjective optimization problem," in *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003)*, vol. 4, Canberra, ACT, Australia: IEEE, 2003, pp. 2292–2297. DOI: 10.1109/CEC.2003.1299374. [Online]. Available: https://doi.org/10.1109/CEC.2003.1299374.

[16] D. Karaboğa and B. Baştürk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007. DOI: 10.1007/s10898-007-9149-x. [Online]. Available: https://doi.org/10.1007/s10898-007-9149-x.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

This research work is an individual work done entirely by the author: Mohammed H.S. Helal.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

This research received no external funding. The work was conducted using equipment provided by Birzeit University. Publication/submission fees, where applicable, were covered by Birzeit University.

Conflicts of Interest

The author declares no conflicts of interest relevant to the content of this article.

APPENDIX

Table 1: Benchmark test functions. *Source: prepared by author.*

Label	Function Name	Definition $f(x)$	Range and Optimum
f_1	Griewank	$f(x) = 1 + \frac{1}{1000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$x_i \in [-600, 600],\ x^* = \mathbf{0}$
f_2	Rastrigin	$f(x) = \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$	$x_i \in [-5.12, 5.12],\ x^* = \mathbf{0}$
f_3	Rosenbrock	$f(x) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	$x_i \in [-30, 30],\ x^* = \mathbf{1}$
f_4	Ackley	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{d}\sum x_i^2}\right) - \exp\left(\frac{1}{d}\sum \cos(2\pi x_i)\right) + 20 + e$	$x_i \in [-32, 32],\ x^* = \mathbf{0}$
f_5	Schwefel 2.21	$f(x) = \max_{1 \le i \le d} \lvert x_i \rvert$	$x_i \in [-100, 100],\ x^* = \mathbf{0}$

Table 2: Suggested parameter settings for the compared optimization algorithms. *Source: The settings parameters for GA are taken from, [15], ABC is from, [16], while PSO, WPA and OGL-WPA are taken from, [12]*

Algorithm	Suggested Parameter Settings
Genetic Algorithm (GA)	Population size $N = 125$ Crossover rat 0.95 Mutation rate 0.1
Particle Swarm Optimization (PSO)	Population size $N = 100$ Inertia weight $w = 0.5$ $c_1 = 0.5, c_2 = 0.5$
Artificial Bee Colony (ABC)	Colony size $N = 125$ Employed:onlooker $=$ 1:1 Scout: random re-initialization
Wolf Pack Algorithm (WPA)	Population size $N = 100$ Leaders: $\alpha = 1000, \beta = 2000, \delta = 3000$
Opposition-learning/GA-based WPA(OGL-WPA)	Population size $N = 100$ $h = 4, T_{max} = 15$ $Step_a = 2, Step_b = 1$ step factor is 1 and $\lambda = 0.5$

Table 3: Results on benchmark functions (mean and standard deviation). *Source: SCO from author's experiments, GA from, [15], ABC from, [16], while PSO, WPA and OGL-WPA are taken from, [12].*

f	Dim	GA, [15]		PSO, [12]		WPA, [12]		ABC, [16]		OGL-WPA, [12]		SCO	
		Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
f_1	5	-	-	0.15087	0.0723825	0.12426	0.03146	-	-	0.00409	0.00402	**6.24E-05**	0.0001608
	10	0.200912	0.118092	0.528817	0.1916648	0.54092	0.10674	0.00348	0.01014	0.00837	0.00405	**0.001446**	0.0054957
	30	4.9368	0.4418	1.029196	0.005423	0.41591	0.15055	**1.15E-08**	3.38E-09	0.00874	0.00886	0.008501	0.02282667
f_2	5	-	-	8.2732804	4.629569	2.987942	1.961736	-	-	0.00021	0.00015	**0**	0
	10	1.3928	0.76319	34.954967	11.06757	17.40869	12.63959	**0**	0	1.72416	1.16982	**0**	0
	30	10.4388	2.6386	201.0472	28.58588	118.3956	69.370107	0.033874	0.181557	11.2121	5.1233	**0.0009787**	0.0007182
f_3	5	-	-	165.48465	360.10007	2.25282	1.485809	-	-	2.449445	5.11299	**0.01343**	0.0369
	10	46.3184	33.8217	8255.2939	13456.5928	9.6771086	1.174687	**0.034072**	0.045553	2.097849	2.25198	0.1521	0.2664
	30	166.283	59.5102	847467.368	634605.3	92.029587	23.22154	**0.219626**	0.152742	34.45741	34.1515	17.96	11.975
f_4	5	-	-	1.9215949	1.1884333	0.0479688	0.027839	-	-	0.00789	0.00293	**0**	0
	10	0.59267	0.22482	6.4053804	1.695019	0.3267526	0.21861631	7.8E-11	1.16E-09	0.01713	0.00427	**0**	0
	30	1.0989	0.24956	12.03379	1.2499723	2.9454203	0.40191428	**3E-12**	5E-12	0.04827	0.00678	0.0008468	0.0009167
f_5	5	-	-	0.8557691	0.88475519	0.05184759	0.02551751	-	-	0.0038199	0.00145466	**0**	0
	10	1.9519	1.3044	10.652086	3.66702843	0.36033598	0.1960764	**1.27E-09**	4E-12	0.0129461	0.0035201	0.0001187	0.0001993
	30	13.5346	4.9534	26.71264	4.98594719	5.1772917	1.12082451	146.8568	82.3144	**0.1228409**	0.03491179	5.353	5.408