Brazilian Journal of Biometrics,**44**,1-14 (2026)
doi:10.28951/bjb.v44i1.925

ARTICLE

C-SOMGOAM: A Self-Organizing Migrating Grasshopper Mutation Algorithm for solving constrained optimization

Dipti Singh[1], Neha Chand*[1]

[1]Department of Applied Mathematics, Gautam Buddha University, Greater Noida, Uttar Pradesh 201312, India.

*Corresponding author. Email: nehachand1980@gmail.com;

(Received: July 10, 2025; Revised: September 1, 2025; Accepted: September 12, 2025; Published: April 29, 2026)
Section editor: João Domingos Scalon.

Abstract
This paper introduces C-SOMGOAM, a self-organizing migrating grasshopper optimization algorithm designed for constrained optimization (CO). The algorithm integrates features from the Grasshopper Optimization Algorithm (GOA), the Self-Organizing Migrating Algorithm (SOMA), and a non-uniform mutation operator. A key contribution of this work is the incorporation of a penalty-free constraint-handling mechanism into GOA to effectively address CO problems. Unlike some traditional methods, the proposed GOA variant explicitly operates on a population of solutions. Starting with a randomly selected population, the algorithm iteratively modifies these solutions to improve the approximation of the global optimum. C-SOMGOAM is not only simple to implement but also capable of generating feasible and high-quality solutions. To assess its performance, the algorithm is evaluated on ten constrained benchmark problems and three engineering design problems from the literature. The effectiveness of C-SOMGOAM is demonstrated through comprehensive result analysis and comparative evaluation against existing variants.

Keywords: Grasshopper optimization algorithm, Self organizing migrating algorithm, Constrained optimization, Penalty-free constraint handling selection.

1. Introduction

Optimization involves finding the most effective combination of decision variables to attain the best possible result within specified constraints. Almost every field of human activity, including engineering, operation research, medicine, finance, computer science and social system etc. demonstrates the need for optimization. Optimization problems are generally classified into two main categories: linear and nonlinear, each of which can be further divided into constrained and unconstrained types. CO plays a major role in solving problems in engineering and industry. Most real-world design optimization problems involve determining the optimal parameters that either maximize or minimize an objective function while satisfying a set of specific requirements known as constraints. This type of optimization problem is generally referred to as COP.

The following is the formulation of a general nonlinear constrained optimization problem:

$$Min/Max\ \ f(x), \qquad x \equiv (x_1, x_2, x_3, \dots \dots . x_n)$$

Subject to:

$$x\epsilon D = \left\{ x\epsilon \mathbb{R}^n \;\middle|\; \begin{array}{ll} g_k(x) \geq 0, & k = 1,2,3, \dots \dots K \\ h_m(x) = 0, & m = \ 1,2,3, \dots \dots M \\ a_i \leq x_i \leq b_i, & i = 1,2,3, \dots \dots n \end{array} \right.$$

where $f(x)$ is the objective function to be minimized or maximized, $g_k(x) \geq 0,\ k = 1,2,3, \dots K$ and $h_m(x) = 0,\ m = 1,2,3, \dots, M$ are inequality and equality constraints, a_i and b_i are lower and upper bounds of the variables, respectively Deep and Singh (2008).

Constrained optimization problems are typically more difficult to solve than their unconstrained counterparts due to the presence of various constraints—such as equalities and inequalities—and their complex interactions with the objective function. These problems often involve nonlinear objective functions that may be continuous, discrete, or mixed, along with nonlinear constraints. There are two general methods used for optimizing such types of problems or functions mathematical programming and meta-heuristic methods. Various conventional mathematical approaches have been employed to solve such types of problems. However, mathematical programming methods have faster convergence rates and higher accuracy in solving COP. These methods operate under the requirement of gradient availability, defined initial points, and a continuous objective function. Yet, due to the complexity of systems, computing complex derivatives and providing suitable initial trial solutions (which also affect iteration size) can be challenging. Therefore, researchers depend on meta-heuristic algorithms to solve COP Garg (2016).

For such cases, evolutionary algorithms offer two main advantages over traditional optimization methods:

1. Evolutionary algorithms do not require strict rules or assumptions, even when the objective function is non-continuous or has many local optima.
2. They are adaptable and capable of handling various types of optimization problems, including those with continuous, integer, or mixed decision variables.

Heuristic algorithms are useful techniques when analytical solutions to optimization issues are either impossible or very difficult to find. Numerous heuristic and meta-heuristic methods have been thoroughly explored and effectively implemented for solving a wide range of constrained and unconstrained nonlinear optimization problems. Many researchers are currently focusing on various heuristic approaches, and they have so far developed a large number of innovative algorithms. Over the past few decades, a wide range of meta-heuristic algorithms have been developed and implemented to solve COP like Genetic Algorithm (GA) Holland (1992), Particle Swarm Optimization (PSO) Eberhart and Kennedy (1995), Self-Organizing Migrating algorithm (SOMA) Singh and Agrawal (2015), Grasshopper Optimization Algorithm (GOA) Saremi *et al.* (2017), Artificial Bee colony Garg (2014), Gravitational search algorithm Rashedi *et al.* (2009) etc. Additionally, local search-based anytime algorithms have recently been proposed to enhance solution quality and efficiency in continuous distributed COP Liao *et al.* (2025)

However, according to the prominent "No Free Lunch" hypothesis Wolpert and Macready (1997), meta-heuristic algorithms may not always provide efficient and effective solutions for every critical problem. While a meta-heuristic may yield good results for a specific design issue, the same strategy might produce poor results for another challenge (Shayanfar and Gharehchopogh, 2018; Yildız *et al.*, 2021) In other words, no single meta-heuristic can offer the best solution for every problem. Hertz and Werra (1987) contended that Tabu Search (TS) proves significantly more effective than Simulated Annealing (SA) in solving graph coloring problems. Conversely, Kuik *et al.* (1993) argued that SA outperforms TS in lot-sizing problems. However, Lee and Kim (1996) discovered that TS and SA were equally efficient in solving a project scheduling problem. When tackling critical problems in engineering fields, meta-heuristics (MHs) have several drawbacks, including slow convergence and the risk of getting trapped in local optima. Additionally,

these issues can lead to higher computational costs. As a result, the development of hybridized, modified, and enhanced MHs is rapidly increasing to combine their strengths and overcome these limitations (Zamani *et al.*, 2022; Yildiz *et al.*, 2022). In the literature, numerous researchers have developed hybrid techniques to address constrained optimization problems characterized by high complexity. Furthermore, much hybridization of these algorithms have been successfully applied to various mathematically modeled real-world problems, including engineering and optimization tasks Kumar *et al.* (2020). Some examples of hybrid optimizers include hybrid grey wolf optimizer Nadimi-Shahraki *et al.* (2021), hybrid artificial bee colony Alatas (2010), and hybrid Particle Swarm Optimization Alatas *et al.* (2009).

One of the most modern optimization algorithms is the Grasshopper Optimization Algorithm (GOA). It is a swarm-based, nature-inspired method that mathematically models and mimics the behavior of grasshopper swarms in the real world. Recently, several improvements to the Grasshopper Optimization Algorithm (GOA) have been explored, including Binary GOA Pinto *et al.* (2019), Chaotic GOA Arora and Anand (2019), Opposition-Based GOA Ewees *et al.* (2018), Hybrid GOA-jDE Jia *et al.* (2019), hybrid Genetic Algorithm-GOA El-Shorbagy and El-Refaey (2020), and Hybrid Gravitational search algorithm-GOA Guo *et al.* (2020). For more detailed information about the GOA and its enhancements, refer to the comprehensive survey conducted by Meraihi *et al.* (2021).

In the literature, the main approaches for solving constrained optimization problems are generally classified into four categories. Penalty function strategy, modifying genetic operator approach, rejecting strategy, and repairing strategy. The most popular of them is the penalty function strategy. The primary disadvantage of this strategy is the requirement for fine-tuning the penalty parameter. Improper handling of this parameter increases the probability of obtaining suboptimal solutions. This article introduces C-SOMGOAM, a unique hybrid approach designed to address nonlinear constrained optimization problems without the need for penalty parameters. It is relatively simple to implement and does not require any parameter tuning for constraint handling. A series of experiments were conducted to evaluate the performance of the hybrid GOA algorithm using ten benchmark problems and three engineering optimization problems—specifically, the Compression Spring Design Problem, pressure vessel design problems and three-bar truss design problems. The experimental results, when compared with those of other similar algorithms, demonstrated that the proposed hybrid variant achieved superior performance and efficiency. The main feature of the proposed GOA versions is that they work with a population of solutions rather than a single solution in each generation. To improve the approximation, they iteratively modify these solutions, starting with a randomly selected population.

The following sections are arranged as follows: Section 2 describes the material and methods. Section 3 details the proposed hybrid approach. Section 4 presents the results and discussion, and Section 5 presents three engineering problems. Finally, Section 6 provides the conclusion.

2. Materials and Methods

This section describes the working principles of SOMA, GOA, and the mutation operator, which form the foundation of the newly proposed variant for constrained problems. It also presents the underlying motivation for integrating these techniques.

2.1 Self Organizing Migrating Algorithm (SOMA)

The Self-Organizing Migrating Algorithm (SOMA) is a population-based, stochastic optimization technique inspired by both competitive and cooperative behaviors among individuals. It operates as a stochastic search method that guides individuals through the search space based on their interactions and relative performance. Although it shares similarities with other evolutionary algorithms, SOMA operates based on a distinct mechanism. Unlike traditional evolutionary algorithms, it does not generate new individuals during the optimization process. Instead, it iteratively moves existing individuals from their current positions toward better solutions in the search space. The movement of an individual is defined by the following equation Singh and Agrawal (2015):

$$X_{i,j}^{MLnew} = X_{i,j,start}^{ML} + (X_{L,j}^{ML} - X_{i,j,start}^{ML}).\text{t.PRTVector j.} \quad (1)$$

where t ∈< 0, by step to, Path Length >, and ML represents the migration loop. $\boldsymbol{X_{i,j}^{MLnew}}$ denotes the new position of an individual. $\boldsymbol{X_{i,j,strt}^{ML}}$ defines the position of the active individual, and $\boldsymbol{X_{L,j}^{ML}}$ represents the position of the leader.

2.2 Basic GOA

The Grasshopper Optimization Algorithm (GOA) is inspired by the swarming behavior of grasshopper (insects). These insects undergo three distinct stages in their life cycle: egg, nymph, and adult. They are considered agricultural pests due to their negative impact on crop productivity. During the nymph stage, grasshoppers primarily exhibit jumping and rolling cylindrical movements characterized by small, slow steps. In this phase, they consume any vegetation encountered in their path. As adults, grasshoppers form large swarms capable of rapid migration over vast distances Ewees *et al.* (2018). All relevant mathematical formulations and equations related to the algorithm are briefly explained in this paper.

The mathematical formulation of grasshopper behavior is described as follows:

$$X_i=S_i+G_i+A_i \tag{2}$$

The Equation 2 can be expressed as follows to provide random behavior:

$$X_i= r_1S_i + r_2G_i + r_3A_i \tag{3}$$

where the value of random number between 0 to 1 and S_i represents the social interaction of the i^{th} grasshopper, which is described as follows:

$$S_i= \sum_{\substack{j=1 \\ j\neq i}}^{N} s(dij)\frac{(Xj-Xi)}{dij} \tag{4}$$

where d_{ij} is the distance between the i^{th} and j^{th} grasshoppers defined as dij= |xj-xi| , the unit vector from grasshopper i^{th} to grasshopper j^{th} is $\widehat{d_{ij}} = \frac{x_j-x_i}{d_{ij}}$,and s is the strength of the social forces function, which is defined as follows.

$$s(r) =f.\, e^{\left(\frac{-r}{l}\right)} - e^{-(r)} \tag{5}$$

Here, f and l denote the strength of attraction and the attractive length scale, respectively. Since the function s approaches zero for distances greater than 10 Saremi *et al.* (2017), it cannot apply significant forces at large separations. To address this limitation, the grasshopper separation distance is normalized to the interval [1, 4]. The G_i and A_i in Equation 2 stand for the i^{th} grasshopper's gravitational force and wind advection, respectively, and are defined as follows:

$$G_i= -g.\, \hat{e}_g, \tag{6}$$

$$A_i=u.\, \widehat{e_w} \tag{7}$$

where $\widehat{e_g}$ and $\widehat{e_w}$ are unit vector toward the earth's center and wind direction, respectively, and g and u for the gravitational constant and a constant drift coefficient.

In Equation 2, the value of S_i,G_i, and A_i from Equations 4,5,6, and 7 were substituted and the updated position of the grasshopper was computed as follows:

$$X_i = \sum_{\substack{j=1 \\ j\neq i}}^{N} s(|x_j - x_i|) \frac{x_j - x_i}{dij} - g\widehat{e_g} + u\hat{e}_w \tag{8}$$

However, Equation 8 above cannot be directly used for optimization. Saremi *et al.* (2017) reformulated the grasshopper position update as follows:

$$X_i^d = c \sum_{\substack{j=1 \\ j\neq i}}^{N} c \frac{u_{bd} - l_{bd}}{2} s\left(\left|x_j{}^d - x_i{}^d\right|\right) \frac{(x_j - x_i)}{dij} + \widehat{Td} \tag{9}$$

$$c = cmax - t\frac{C_{max} - C_{min}}{T} \tag{10}$$

where u_{bd} and l_{bd} represent the upper and lower bounds in the dth dimension, respectively, and $\widehat{Td}$ denotes the optimal solution found so far in the dth dimension. Note that S refers to the same component used in the equation, where G is equal to zero and A is always directed toward the best solution $\widehat{Td}$. The first parameter c in the Equation 9 is analogous to the inertia weight w in Particle Swarm Optimization (PSO); it helps control the movement of grasshoppers around the food source and provides a good balance between exploration and exploitation. The second parameter c reduces the influence of repulsion, attraction, and comfort zones among grasshoppers as the number of iterations progresses. Here, c_{max} and c_{min} are the maximum and minimum values of c, respectively. The variable t denotes the current iteration, and T represents the total number of iterations. Each grasshopper updates its position based on its current location, the global best position, and the positions of other grasshoppers in the swarm. This mechanism helps GOA avoid being trapped in local optima Maraihi *et al.* (2021). The pseudo-code for the standard Grasshopper Optimization Algorithm is presented in Algorithm 1.

Algorithm 1 Grasshopper Optimization Algorithm (GOA) Maraihi *et al.* (2021)

1. Initialize the population P_i for i= 1, 2, …..,n
2. Initialize control parameters: c_{min}, c_{max} ,maximum iterations T
3. Compute the fitness value $f(P_i)$of all grasshoppers P_i
4. Set P as the best solution found so far
5. **While** t < T **do**
6. Evaluate control parameter c using Equation 10
7. **for** every grasshopper i=1 to n **do**
8. Normalize the separation between grasshoppers to the interval [1, 4]
9. Calculate the next position of grasshopper P_i according to Equation 9
10. If the updated position is outside the search space, return the grasshopper to the boundary
end for
11. **Update** the best solution P if a better one is found.
12. **end While**
13. Return best solution
14. **Stop**

2.3 Non Uniform Mutation Operator

One solution, x_{ij}, is randomly selected and its value is updated according to the following rule Singh and Agrawal (2014):

$$x_{ij}^{0} = \begin{cases} x_{ij} + \left(u_{bj} - x_{ij}\right)\tau(t) & if\ a_1 < 0.5, \\ x_{ij} - \left(x_{ij} - l_{bj}\right)\tau(t) & if\ a_1 \geq 0.5 \end{cases} \tag{11}$$

where a_1 and a_2 two random values in the interval [0, 1] and b is a constant parameter, t represents the current iteration number, and t_{max} is the maximum number of iterations the algorithm can execute. The lower bound $\left(l_{bj}\right)$ and upper bound $\left(u_{bj}\right)$ correspond to the bounds of the problem's parameters that are to be optimized. The value of $\tau(t)$ is defined as

$$\tau(t) = \left(a_2\left(1 - \frac{t}{t_{max}}\right)\right)^{b}$$

3. Methodology of C-SOMGOAM

Sometimes, the Grasshopper Optimization Algorithm (GOA) performs well in solving unconstrained optimization problems. However, it faces difficulties when dealing with problems that involve strict constraints. Addressing these constraints is the main challenge in solving constrained optimization problems. There are two types of constraints in such problems: equality constraints and inequality constraints. These make it difficult to determine feasible solutions.

Numerous strategies have been proposed to address these limitations, with penalty functions being the most commonly used approach in the field of evolutionary algorithms. The use of a penalty function discourages infeasible solutions by reducing their fitness based on how much they violate constraints, thus favoring feasible alternatives. Although widely used, penalty functions have several disadvantages. The primary drawback is the need to select numerous parameters, making it difficult to find an appropriate combination. Additionally, the search process tends to be slow, and there is no guarantee that the optimal solution will be found. To overcome these issues, this article introduces a penalty-free hybrid approach combined with a constraint-handling strategy. In particular, we present a hybrid Self-Organizing Migrating Grasshopper Mutation Algorithm (C-SOMGOAM), which integrates GOA with the Self-Organizing Migrating Algorithm (SOMA) and a non-uniform mutation operator to enhance performance on constrained optimization problems.

The following is a mathematical formula for the constraint violation function:

$$\psi(x) = \sum_{m=1}^{M}[h_m(x)]^2 + \sum_{m=1}^{K} G_k[g_k(x)]^2, \tag{12}$$

where $h_m(x)$ represent the equality constraints and $g_k(x)$ represents the inequality constraints, and G_k denotes the Heaviside operator, defined as

$$G_k = \begin{cases} 0, & g_k(x) \geq 0, \\ 1, & g_k(x) < 0. \end{cases}$$

In this, solutions within the feasible region have a value of constraint violation function zero. Conversely, solutions outside the feasible region have a non-zero constraint violation function value, signifying their distance from feasibility. The suggested approach randomly generates a population of feasible points and evaluates them using a fitness function. The positions of these points are updated by GOA to yield a potential candidate for the global optimum. The grasshopper moves to the new position if it is superior to the current one; otherwise, it stays in its existing position. Following this, the fittest grasshopper is selected as the leader of each generation, while the least fit is designated as the active grasshopper. The active grasshopper then moves toward the leader (Path length/step size). For active grasshopper, a new position is generated. This population consists simply of the new positions of the active grasshopper. Equation 1 defines the active individual's movement. The population is then sorted based on fitness values. The evaluation starts from the best position based on the constraint violation function of the new population. If $\psi(x) = 0$, the active grasshopper is replaced with its current position. If $\psi(x) > 0$, the algorithm moves to the next best position in the newly sorted population. If no feasible solution is found, the active individual remains unchanged. Once again, the best and worst grasshoppers in the population are selected. In this phase, a new point is generated using this operator by Equation 11. This point is accepted only if it is better than the active individual and satisfies the constraint violation function, in which case it replaces the active grasshopper. This process is repeated until the termination criterion is met.

The computational complexity of C-SOMGOAM grows linearly with the population size, problem dimension, and number of iterations. Fitness evaluation dominates the cost, while position updates add a smaller linear component and the mutation operator contributes only negligible overhead. Overall, the algorithm is as efficient as other population-based meta-heuristics, with the additional advantage that the penalty-free constraint-handling mechanism avoids extra parameter tuning. The pseudo-code of the proposed C-SOMGOAM is shown in Algorithm 2, and its working steps are illustrated in the flow chart given in Figure 1.

Algorithm 2 Constrained C-SOMGOAM Algorithm

1. **Initialize** parameters: C_{min}, C_{max} ,maximum iterations T, path length, Step size, PRT parameter, Population size, and search bounds
2. Randomly generate initial feasible grasshopper positions
3. Compute the fitness value of each grasshopper
4. **While** stopping criteria not satisfied **do**
5. Update the positions of grasshoppers using Equation 9
6. Choose the best grasshopper as the leader and the worst as the active one
7. Sort all grasshoppers based on their fitness values
8. Generate the PRT vector and compute new grasshopper positions using Equation 1
9. **If** feasibility condition is satisfied **then**
   Replace the active grasshopper with one of the feasible positions
10. **else**
    Apply non-uniform mutation using Equation 11 to generate a new position for the grasshopper
    If feasibility condition is satisfied **then**
    Update the active grasshopper's position with the mutated one
    end If
11. **end If**
12. **end While**
13. **Stop**

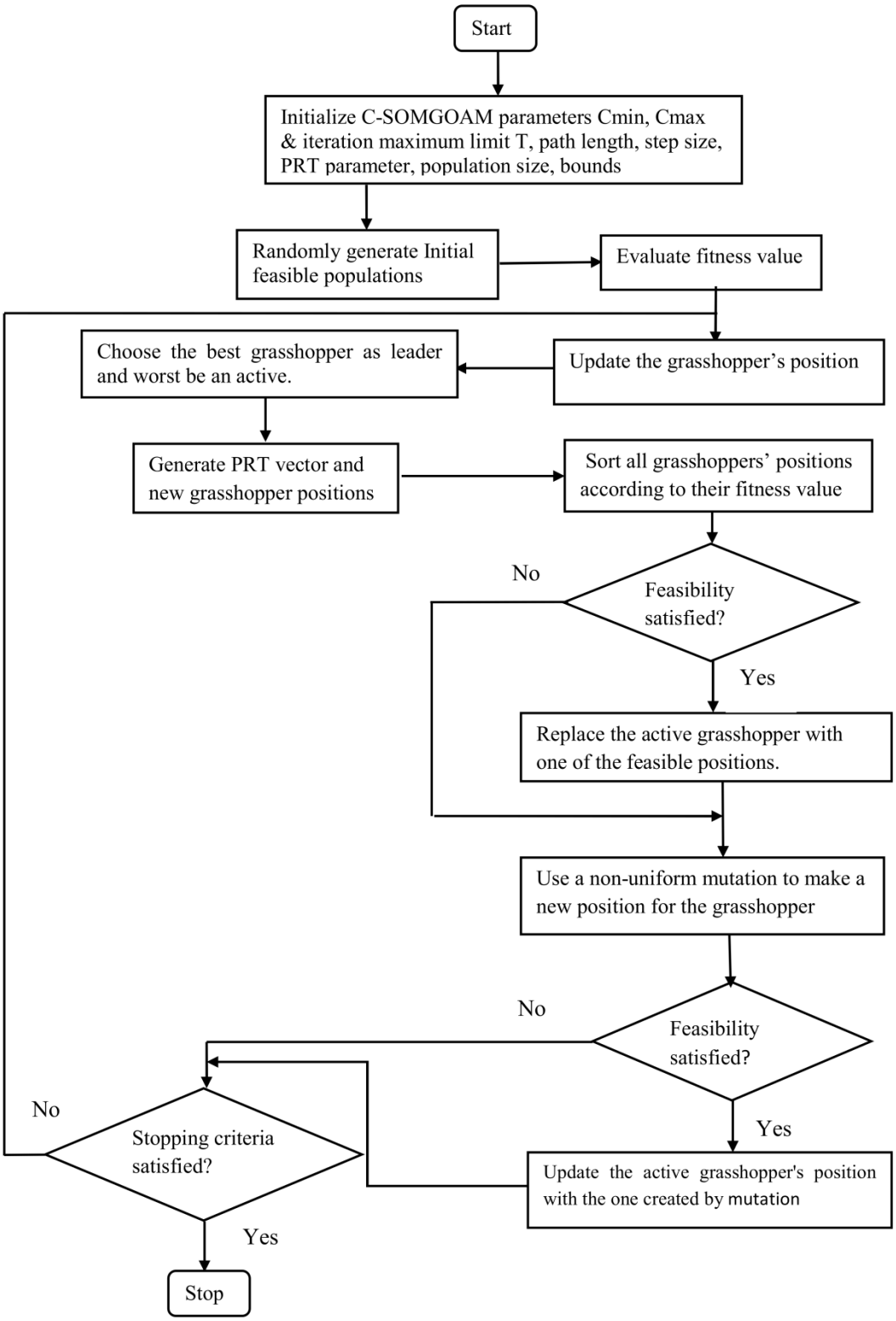

Figure 1. Flowchart for proposed hybrid variant C-SOMGOAM.

4 Results and Discussion

Ten benchmark test functions and three engineering design problems from the literature were used to evaluate the effectiveness of the proposed C-SOMGOAM algorithm. These benchmark problems involve objective functions defined over decision variables and are subject to both equality and inequality constraints. For consistency, any maximization problems were converted into minimization problems by transforming the objective function $f(x)$ to $-f(x)$. The ten benchmark functions, widely used in previous studies, were adopted from existing literature Singh *et al.* (2016). Since C-SOMGOAM is a probabilistic algorithm, each test was executed 30 times. A run is considered successful if it achieves an approximate optimal solution; otherwise, it terminates upon reaching the maximum number of iterations. N/A indicates that no near-optimal solution was found. Table 1 presents the parameter settings used for C-SOMGOAM, while Table 2 compares its performance with other algorithms, including C-GA, C-SOMA, C-GOA, and another GOA variant, C-SOMGOA. Tables 3, 4, and 5 present the best solutions obtained for the three engineering problems.

Unlike traditional penalty-based approaches, the proposed penalty-free constraint-handling strategy further strengthens these results. It eliminates the need for tuning penalty parameters, thereby simplifying implementation and improving robustness across diverse problem types. Moreover, feasibility is enforced directly through the constraint violation function, which ensures a balanced trade-off between exploration of the search space and exploitation of feasible regions without bias from penalty weights.

Table 1. parameters of C-SOMGOAM

Control Parameter	Fine tuned value
Population size	20
Step size	0.05
Path length	2
PRT	0.1, 0.3
Maximum iteration	1000

Table 2. Average objective function values after 1000 maximum iterations

No.	C-GA	C-SOMA	C-GOA	C-SOMGOA	C-SOMGOAM
1	-1.82244	-1.87228	-1.86934	-1.86990	**-1.87190**
2	316.76590	311.60180	315.900	317.911	**319.228**
3	0.02241	0.26864	0.90120	0.05431	**0.01807**
4	202.46040	-234.96490	-298.460	-302.150	**-310.00**
5	94.43180	6.73079	9.0610	**13.5920**	**13.5920**
6	-13.23778	-230.5195	N/A	**-14.890**	**-14.890**
7	0.05470	-7.09413	N/A	-0.08910	**-0.09556**
8	0.80152	0.14079	0.81078	**0.749912**	**0.749912**
9	0.46882	-0.23646	-0.7724	-0.97100	**-1.00956**
10	0.34575	0.24275	-0.8002	**-0.803374**	**-0.80337**

5 Constrained engineering optimization problems

Traditional optimization algorithms often struggle to effectively solve complex engineering design problems. This section presents three well-known engineering optimization problems that were solved using the newly proposed hybrid variant, demonstrating its effectiveness in enhancing performance over traditional methods.

5.1 Compression Spring Design Problem

The design of the tension/compression spring is detailed in Ewees *et al.* (2018). The primary objective of this problem is to minimize the weight of the spring while satisfying constraints related to shear stress, surge frequency, deflection, outer diameter, and bounds on the design variables. The three key design variables are the wire diameter (d), mean coil diameter (D), and number of active coils (N). The mathematical formulation of the problem as follows:

Minimize $f(x_1, x_2, x_3) = (x_3 + 2)x_2x_1^2$

Subject to $g_1(x) = 1 - \frac{x_2^3 x_3}{71785x_1^4} \leq 0,$

$$g_2(x) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leq 0,$$

$$g_3(x) = 1 - \frac{140.45x_1}{x_2^2x_3} \leq 0\ ,$$

$$g_4(x) = \frac{x_2 + x_1}{1.5} - 1 \leq 0,$$

$0.05 \leq x_1 \leq 2, \quad 0.25 \leq x_2 \leq 1.3, \quad 2.0 \leq x_3 \leq 15,$

where the design vector is defined as:

$$X = (\boldsymbol{d}, \boldsymbol{D}, \boldsymbol{N}) = (\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3})$$

Table 3. Best solution of compression Spring Design Problem

Algorithms	d	D	N	Best optimal solution
GA Coello (2000)	0.05148	0.351661	11.632201	0.01270478
MVO Ewees et al (2018)	0.05251	0.37602	10.335130	0.01279000
WOA Ewees et al (2018)	0.051207	0.345215	12.004032	0.01267630
GSA Ewees et al (2018)	0.050276	0.323680	13.525410	0.01270220
C-SOMAQI	0.05175	0.358183	11.230800	0.01266557
C-SOMGOA	0.051770	0.35920	11.191000	0.01269903
C-SOMGOAM	0.0517991	0.34991	11.489900	0.01266519

5.2 Pressure Vessel Design Problem

In this problem, a cylindrical pressure vessel is considered, which is capped with hemispherical heads at both ends and joined by two longitudinal welds to form a complete structure. The objective is to minimize the total cost of the pressure vessel by optimizing four decision variables: the thickness of the vessel ($\boldsymbol{T_s}$), the thickness of the head ($\boldsymbol{T_h}$), the inner radius of the vessel ($\boldsymbol{R}$), and the length of the cylindrical section excluding the heads ($\boldsymbol{L}$) Ewees et al (2018).

Minimize $f(x_1, x_2, x_3, x_4) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$

Subject to $g_1(x) = -x_1 + 0.0193x_3 \leq 0,$

$$g_2(x) = -x_2 + 0.00954x_3 \leq 0,$$

$$g_3(x) = -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0\,,$$

$$g_4(x) = x_4 - 240 \leq 0,$$

$0 \leq x_i \leq 100$, i=1, 2: $10 \leq x_i \leq 200$, i= 3, 4.

Where the design vector is defined as:

$$X = [\boldsymbol{T_s}, \boldsymbol{T_h}, \boldsymbol{R}, \boldsymbol{L}] = [\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, \boldsymbol{x_4}]$$

Table 4. Best solution of pressure Vessel Design Problem

Algorithms	T_h	T_s	R	L	Best optimal solution
GA Coello (2000)	0.8125	0.4375	42.097398	176.65405	6059.94634
GSA Ewees *et al.* (2018)	1.1250	0.6250	55.9886598	84.4542025	8538.8359
C-SOMAQI	0.777408	0.38458	40.3123	200	5882.92
C-SOMGOA	0.777408	0.384273	40.2802	200	5872.27388299
C-SOMGOAM	0.777408	0.38399	40.2802	200	5871.483404

5.3 Three-bar truss design problem

The three-bar planar truss design problem was initially proposed by Nowcki (1974) and later revisited by Rai and Saini (2001). It is a constrained non-linear fractional programming problem. The objective is to minimize the volume of a statically loaded three-bar truss, subject to stress (σ) constraints on each truss member. The design variables are the optimal cross-sectional areas x_1 and x_2, which must be determined to satisfy the given constraints Ewees *et al.* (2018).

$$Minimize\ f(x_1, x_2) = 2(\sqrt{2x_1 + x_2}) \times l$$

$$subject\ to \quad g_1(x) = \left(\frac{\sqrt{2x_1} + x_2}{\sqrt{2x_1^2 + 2x_1x_2}}\right)P - \sigma \leq 0,$$

$$g_2(x) = \left(\frac{x_2}{\sqrt{2x_1^2 + 2x_1x_2}}\right)P - \sigma \leq 0,$$

$$g_3(x) = \left(\frac{1}{x_1 + \sqrt{2x_2}}\right)P - \sigma \leq 0,$$

$$0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1,$$

Where: l=100cm, $P = 2KN/cm^2$, $\sigma = 2KN/cm^2$

Table 5 Best solution of Three-bar truss design problem

Algorithms	X_1	X_2	Best optimal solution
GOA Ewees *et al.* (2018)	0.7888975	0.40761957	263.8958815
OBLGOA Ewees *et al.* (2018)	0.78866365	0.40828079	263.8958440
C-SOMAQI	0.788288	0.409343	263.8958000
C-SOMGOA	0.789001	0.40717	263.8951000
C-SOMGOAM	0.789001	0.40717	263.8951000

6 Conclusions

This paper presents C-SOMGOAM, a hybrid algorithm that integrates the Grasshopper Optimization Algorithm (GOA) with SOMA and a mutation operator to solve constrained nonlinear optimization problems. The proposed method employs a penalty-parameter-free strategy and achieves efficient performance with a relatively small population size. As a GOA variant, C-SOMGOAM is compared against C-GA, C-SOMA, C-GOA, and other SOMA-based algorithms. Unlike traditional GOA implementations, it operates on a population of solutions, enhancing exploration and exploitation capabilities. The algorithm's simplicity, ability to generate feasible solutions, and superior performance were validated through experiments on ten benchmark problems and three real-world engineering design cases. Comparative analysis confirms its effectiveness and robustness, positioning C-SOMGOAM as a promising tool for constrained optimization. In the future, this variant can be extended to solve more complex constrained and engineering problems.

Acknowledgments

The authors acknowledge the support provided by the Department of Applied Mathematics, Gautam Buddha University for conducting this research. Authors also thank reviewers and editor for their thoughtful comments.

Conflicts of Interest

The authors state that there are no competing interests.

Funding

This study did not receive any external funding.

Data availability

The data used in this paper were generated through experimental runs.

Author Contributions

Conceptualization: SINGH, D.; CHAND, N.; **Data curation:** SINGH, D.; CHAND, N.; **Formal analysis:** SINGH, D.; CHAND, N.; **Investigation:** SINGH, D.; **Methodology:** SINGH, D.; CHAND, N.; **Software:** CHAND, N.; **Supervision:** SINGH, D.; **Validation:** SINGH, D.; CHAND, N.; **Visualization:** SINGH, D.; CHAND, N.; **Funding Acquisition:** SINGH, D.; CHAND, N.; **Writing - original draft:** CHAND, N.; **Writing - review and editing:** SINGH, D.

References

1. Arora, S., Anand, P. Chaotic grasshopper optimization algorithm for global optimization. *Neural Computing and Applications* **31**, 4385–4405 (2019). https://doi.org/10.1007/s00521-018-3343-2
2. Alatas, B., Akin, E., Ozer, A.B. Chaos embedded particle swarm optimization algorithms. Chaos, *Solitons & Fractals* **40**, 1715–1734 (2009). https://doi.org/10.1016/j.chaos.2007.09.063
3. Alatas, B.: Chaotic bee colony algorithms for global numerical optimization. Expert *systems with applications* **37**, 5682–5687 (2010). https://doi.org/10.1016/j.eswa.2010.02.042
4. Coello, C.A.C. Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry* **41**, 113–127 (2000). https://doi.org/10.1016/S0166-3615(99)00046-9
5. Deep, K., & Singh, D. A self-organizing migrating genetic algorithm for constrained optimization. *Applied Mathematics and Computation* **198**, 237–250 (2008). https://doi.org/10.1016/j.amc.2007.08.032
6. Ewees, A. A., Abd Elaziz, M., Houssein, E. H. Improved grasshopper optimization algorithm using opposition-based learning. *Expert Systems with Applications* **112**, 156–172 (2018). https://doi.org/10.1016/j.eswa.2018.06.023
7. Eberhart, R., Kennedy, J. A new optimizer using particle swarm theory. In: MHS'95. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp.39–43 (1995). Ieee. https://doi.org/10.1109/MHS.1995.494215
8. El-Shorbagy, M.A., El-Refaey, A.M.: Hybridization of grasshopper optimization algorithm with genetic algorithm for solving system of non-linear equations. *IEEE Access* **8**, 220944–220961 (2020). https://doi.org/10.1109/ACCESS.2020.3043029
9. Garg, H. Solving structural engineering design optimization problems using an artificial bee colony algorithm. *J Ind Manag Optim* **10**, 777–794 (2014). https://doi.org/10.3934/jimo.2014.10.777
10. Garg, H.: A hybrid pso-ga algorithm for constrained optimization problems. Applied Mathe-matics and Computation 274, 292–305 (2016). https://doi.org/10.1016/j.amc.2015.11.001
11. Guo, S.-S., Wang, J.-S., Xie, W., Guo, M., Zhu, L.-F. Improved grasshopper algorithm based on gravity search operator and pigeon colony landmark operator. *IEEE Access* **8**, 22203–22224 (2020). https://doi.org/10.1109/ACCESS.2020.2967399
12. Holland, J. H. Genetic algorithms. *Scientific american* **267**, 66–73 (1992). https://doi.org/10.1038/scientificamerican0792-66
13. Hertz, A., Werra, D. D. Using tabu search techniques for graph coloring. *Computing* **39**, 345–351 (1987). https://doi.org/10.1007/BF02239976
14. Jia, H., Li, Y., Lang, C., Peng, X., Sun, K., Li, J. Hybrid grasshopper optimization algorithm and differential evolution for global optimization. *Journal of Intelligent & Fuzzy Systems* **37**, 6899–6910 (2019). https://doi.org/10.3233/JIFS-190782
15. Kumar, N., Mahato, S. K., Bhunia, A. K. A new qpso based hybrid algorithm for constrained optimization problems via tournamenting process. *Soft Computing* **24**, 11365–11379 (2020). https://doi.org/10.1007/s00500-019-04601-3
16. Kuik, R., Salomon, M., Van Wassenhove, L. N., Maes, J. Linear programming, simulated annealing and tabu search heuristics for lotsizing in bottleneck assembly systems. *IIE transactions* **25**, 62–72 (1993).https://doi.org/10.1080/07408179308964266
17. Lee, J.-K., Kim, Y.-D. Search heuristics for resource constrained project scheduling. *Journal of the Operational Research Society* **47**, 678–689 (1996). https://doi.org/10.1057/jors.1996.79
18. Meraihi, Y., Gabis, A. B., Mirjalili, S., Ramdane-Cherif, A. Grasshopper optimization algorithm: theory, variants, and applications. *Ieee Access* **9**, 50001–50024 (2021). https://doi.org/10.1109/ACCESS.2021.3067597
19. Nadimi-Shahraki, M. H., Taghian, S., Mirjalili, S. An improved grey wolf optimizer for solving engineering problems. *Expert Systems with Applications* **166**, 113917 (2021). https://doi.org/10.1016/j.eswa.2020.113917

20. Pinto, H., Peña, A., Valenzuela, M., Fernández, A. A binary grasshopper algorithm applied to the knapsack problem. In: Artificial Intelligence and Algorithms in Intelligent Systems. *Proceedings of 7th Computer Science On-line Conference 2018*, **27**, 132–143 (2019). Springer. https://doi.org/10.1007/978-3-319-91189-2_14
21. Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S. Gsa: a gravitational search algorithm. *Information sciences* **179**, 2232–2248 (2009). https://doi.org/10.1016/j.ins.2009.03.004
22. Singh, D., Agrawal, S. A novel hybrid self organizing migrating algorithm with mutation for global optimization. *Int J Soft Comput Eng* **3**, 101–106 (2014). https://www.ijsce.org/portfolio-item/f2006013614/
23. Singh, D., Agrawal, S. Log-logistic soma with quadratic approximation crossover. In: *International Conference on Computing, Communication & Automation*, pp. 146–151 (2015). https://doi.org/10.1109/CCAA.2015.7148380
24. Singh, D., Agrawal, S., Deep, K. C-somaqi: self organizing migrating algorithm with quadratic interpolation crossover operator for constrained global optimization. *Self-Organizing Migrating Algorithm: Methodology and Implementation* 147–165 (2016). https://doi.org/10.1007/978-3-319-28161-2_7
25. Shayanfar, H., Gharehchopogh, F. S. Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems. *Applied Soft Computing* **71**, 728–746 (2018). https://doi.org/10.1016/j.asoc.2018.07.033
26. Saremi, S., Mirjalili, S., Lewis, A. Grasshopper optimisation algorithm: theory and application. *Advances in engineering software* **105**, 30–47 (2017). https://doi.org/10.1016/j.advengsoft.2017.01.004
27. Wolpert, D. H., Macready, W. G. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation* 1, 67–82 (1997). https://doi.org/10.1109/4235.585893
28. Yildiz, B. S., Mehta, P., Sait, S. M., Panagant, N., Kumar, S., Yildiz, A. R. A new hybrid artificial hummingbird-simulated annealing algorithm to solve constrained mechanical engineering problems. *Materials Testing* **64**, 1043–1050 (2022). https://doi.org/10.1515/mt-2022-0123
29. Yildiz, B. S., Patel, V., Pholdee, N., Sait, S. M., Bureerat, S., Yildiz, A. R. Conceptual comparison of the ecogeography-based algorithm, equilibrium algorithm, marine predators algorithm and slime mold algorithm for optimal product design. *Materials Testing* **63**,336–340 (2021). https://doi.org/10.1515/mt-2020-0049
30. Zamani, H., Nadimi-Shahraki, M. H., Gandomi, A. H. Starling murmuration optimizer: A novel bio-inspired algorithm for global and engineering optimization. *Computer Methods in Applied Mechanics and Engineering* **392**, 114616 (2022). https://doi.org/10.1016/j.cma.2022.114616
31. Liao, X., Hoang, K., & Luo, X. Local search-based anytime algorithms for continuous distributed constraint optimization problems. *IEEE/CAA Journal of Automatica Sinica* **12**, 288-290 (2025). https://doi.org/10.1109/JAS.2024.124413