machine learning & knowledge extraction

MDPI

Article

Boolean Particle Swarm Optimization with 0-Mutation

Zaharias D. Zaharis [1], Emmanouil Georgios Nikopolitidis [2], Pavlos I. Lazaridis [3], Panagiotis Sarigiannidis [4] and Sotirios K. Goudos [5,*]

[1] ELEDIA@AUTH, School of Electrical and Computer Engineering, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece; zaharis@auth.gr
[2] School of Electrical and Computer Engineering, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece; nikopole@ece.auth.gr
[3] School of Computing and Engineering, University of Huddersfield, Huddersfield HD1 3DH, UK; p.lazaridis@hud.ac.uk
[4] Department of Electrical and Computer Engineering, University of Western Macedonia, 50100 Kozani, Greece; psarigiannidis@uowm.gr
[5] ELEDIA@AUTH, School of Physics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
* Correspondence: sgoudo@physics.auth.gr

Abstract

First introduced in 1995, the Particle Swarm Optimization (PSO) algorithm offers a reliable and efficient solution to real-valued optimization problems. However, extending it to binary-valued problems proved challenging. This paper proposes a new Boolean version of the PSO technique based on a novel mutation strategy. By employing an innovative mutation mechanism in the velocity bitstring, the method enforces a minimum perturbation level, reduces the risk of premature convergence, and promotes broader global search. Several variations of the algorithm and parameter combinations are evaluated using 47 benchmark functions to derive the best-performing configuration, which is then compared with other population-based methods to demonstrate the effectiveness of the proposed algorithm. Finally, the technique is applied to an antenna array thinning problem for the design of a planar antenna array with certain specifications.

Keywords: particle swarm optimization; binary PSO; boolean PSO; mutation; antenna design; array thinning

Check for updates

Academic Editor: Fangfang Zhang

Received: 30 March 2026
Revised: 26 April 2026
Accepted: 27 April 2026
Published: 3 May 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Particle swarm optimization (PSO) was first proposed by Kennedy and Eberhart [1] and has since attracted considerable attention due to its simplicity, ease of implementation, computational efficiency, flexibility, and overall effectiveness. It is an optimization technique inspired by the movement of groups of animals (e.g., flocks of birds, schools of fish) during their search for food. Initially, the swarm does not have any information about the location of the food, so its members move randomly. In these random movements, each individual locates quantities of food without, however, stopping the search in other locations. Each individual remembers the location of their largest food find, while communication among members ensures the entire group knows the overall best location found. Thus, the next move of each member will be influenced by both their own best experience and the overall knowledge of the swarm.

Kennedy and Eberhart also presented a discrete binary variant of PSO for discrete optimization problems [2], which we will denote BinPSO hereafter. In BinPSO, each particle's position is represented by binary values, specifically 0 or 1. Each position component can

change from 0 to 1 or vice versa. In BinPSO, the velocity of a particle is interpreted as the likelihood that a position component will change its state.

The mathematical modeling of this natural behavior has led to an effective optimization algorithm designed for real-valued problems. However, when it comes to discrete-value encoding, the effectiveness of the corresponding algorithms is still subject to improvement. The primary contribution of this work, analyzed in detail in Section 3.3, is the introduction and systematic validation of a novel 0-mutation strategy in Boolean Particle Swarm Optimization (BPSO), where velocity zeros are selectively converted to ones, fundamentally redefining how exploration is enforced in discrete PSO. Unlike existing BPSO variants (Afshinmanesh et al. [3]) that restrict velocity through $1 \rightarrow 0$ mutation—often leading to premature convergence and stagnation—we demonstrate both analytically and experimentally that promoting controlled $0 \rightarrow 1$ transitions sustains meaningful bit-level perturbations in the XOR-based position update mechanism. Beyond proposing this mutation mechanism, we rigorously evaluate multiple mutation–inertia scheduling strategies across 47 benchmark functions and statistically validate performance, showing that the linear mutation–linear inertia variant (BPSO-LM-LI) consistently achieves superior or statistically competitive results. Furthermore, we follow the same process to establish the proposed 0-mutation BPSO against existing optimization algorithms such as BinPSO, a Binary Grey Wolf Optimizer, a Binary Genetic Algorithm, and a Binary Differential Evolution Algorithm. Importantly, we move beyond synthetic benchmarks and demonstrate the practical potential of the approach by solving a high-dimensional planar antenna array thinning problem (16×16 array), achieving significant element reduction (≈62–64% active elements) while maintaining high directivity and near-target side-lobe level (*SLL*) constraints. This establishes the proposed 0-mutation BPSO not merely as a variation, but as a structurally improved Boolean search mechanism with demonstrated robustness, statistical significance, and real-world applicability.

After presenting the state of the art in Section 2, this paper describes the fundamental logic and equations of the PSO algorithm in Section 3. Then, in Section 3.2, it presents an existing version of the PSO method designed for problems with Boolean variables, while the new method is introduced in Section 3.3. In Section 4, different combinations of PSO parameters are tested alongside the new technique to discover the optimal parameter set and highlight the influence of parameter tuning on the efficiency of the algorithm. Section 4.2 is dedicated to comparing the proposed technique with other existing algorithms. In addition, in Section 4.4, the BPSO-LM-LI with 0-mutation is applied to a real-world antenna-thinning problem to prove its effectiveness. Finally, Sections 5 and 6 present the discussion and conclusions.

2. State of the Art

Shami et al. [4] frame PSO as a widely used swarm optimizer that remains the subject of active modification; the standard formulation still suffers significantly from premature convergence, which motivates a large number of variants. They explicitly categorize the dominant modification strategies into four directions: (i) controlling-parameter modifications, (ii) hybridization with other metaheuristics, (iii) cooperation, and (iv) multi-swarm techniques. They highlight the rising interest in the algorithm by noting the increasing volume of publications after 2017 (representing over 45% of total publications). Ref. Zhang et al. [5] also reports that PSO has accumulated more publications than other Swarm Intelligence (SI) based methods—such as ACO, DE, ABC, BA, and BFO—indicating its dominant position in the field.

As a result, there is extensive literature covering every aspect of the algorithm. More precisely, population size has been found to be highly relevant to the objective function; it

must be capable of exploring a sufficient portion of the feasible area to avoid being trapped in local minima. Generally, a fixed size between 20 and 50 particles is used [6]. Larger populations are usually connected with better performance, but also with higher computational cost. This issue lies at the center of [7], where the authors propose a dynamic-population Multi-Objective PSO (D-MOPSO), in which the population size changes according to the state of the archive and the non-dominated solutions found during the search.

Parameters c_1 and c_2 are usually set around 2.0 [6], but there are also works proposing dynamic tuning of these values during the optimization process, such as [8], where parameter selection is linked to the evolution of particle positions.

The inertia weight ω, which controls the contribution of the previous velocity to the current velocity update, plays a key role in the algorithm. Regarding inertia weight techniques, several strategies have been proposed for their design. Constant and linearly decreasing schemes are among the most common. Other approaches include exponential dynamic [9] and sigmoid increasing [10] inertia weights. In [11], the authors show experimentally that, if properly parameterized, simple linear decreasing inertia weight should not necessarily be considered inferior to other dynamic inertia strategies such as chaotic [12] and random [13].

Neighborhood topologies represent another significant modification of PSO, as they affect communication between particles and, consequently, the search dynamics. While the original PSO is based on a star topology, in which each particle considers all others as neighbors, other alternatives have also been found useful. In [14], three topologies are investigated: one in which each particle interacts with its previous or next neighbor, one in which it communicates with both the previous and next neighbors, and a stochastic scheme. The particle ordering is defined by their indices. The two-sided neighborhood is reported to be more effective for complex unimodal functions, whereas all three topologies achieve good performance in multimodal functions. The issue of choosing the best topology is further addressed in [15], where the authors, after extensive analysis, conclude that topology selection is not only problem-driven but also computational-budget-driven, and provide two formulas for estimating the optimal topology parameters.

In addition, other techniques have been utilized to enhance PSO effectiveness. In [16], smaller swarms are repeatedly formed from the main population according to regrouping schedules, while information exchange between them is also enabled. The multi-swarm approach has also been combined with parallel implementations; for example, in [17], subswarms are executed in CUDA streams, and the proposed framework achieves substantial speedup over the serial version. Negative PSO (NPSO), presented in [18], is another interesting technique aimed at avoiding poor regions of the search space, in which particles avoid their own and the group's previous worst positions rather than being attracted to the best recorded positions.

A major evolutionary direction in PSO research has been its hybridization with other population-based algorithms, such as Genetic Algorithms (GAs) and Grey Wolf Optimizer (GWO). Authors in [19] propose a hybrid PSO-GA algorithm that leverages GA operators, such as crossover and mutation, to modify the current set of solutions and improve the exploration–exploitation balance. Furthermore, another population-based algorithm, the Grey Wolf Optimizer, is combined with PSO in [20], where the proposed hybrid is reported to achieve better performance than the original algorithms.

Regarding discrete search spaces, since Kennedy and Eberhart introduced BinPSO [2], attempts to modify and enhance its performance have been more limited compared to continuous PSO [4]. Binary Particle Swarm Optimization (BPSO) extends the standard framework to spaces where positions take binary values. In BPSO, velocity departs from its original physical interpretation and is instead used to express the probability of a bit flip. The continuous velocity is usually mapped to the interval $[0, 1]$ through a transfer function,

most often the sigmoid, and the position is then updated probabilistically [4]. This inherent randomness can cause BPSO to struggle with local-minima entrapment and inertia-weight setting, according to [21].

To address these issues, several variants have been proposed. Enhanced BPSO (E-BPSO) is introduced in [22] and tries to avoid sigmoid saturation by using a threshold rule on $x + v$ to update the position bits rather than the standard sigmoid-based rule. Another variation is the Multi-phase Discrete PSO (M-DiPSO) proposed in [23], where particles cycle between phases based on their previous state and the number of iterations remaining until the end of the algorithm. Within each phase, the particles are divided into different groups with different parameterizations. Another variation, the BPSO-EPD, avoids local optima by boosting exploration through population-level selection; the worst half of the population is discarded and regenerated around elite individuals [24]. Furthermore, hybridization with the Sine–Cosine Algorithm (SCA) has been explored to enhance feature selection [25].

Nevertheless, BPSO remains highly sensitive to transfer function design. Recent work has focused on redesigning these functions, introducing, apart from the conventional S-shaped ones, V-shaped [26] and Z-shaped transfer functions [27] to stabilize convergence. Finally, a combined V-shaped and U-shaped transfer function aimed at balancing exploration and exploitation [28], as well as time-varying transfer functions [29], have also been proposed in the literature.

Antenna array synthesis and thinning constitute well-established and computationally demanding applications of these techniques. Ref. Boeringer and Werner [30] demonstrates that PSO can effectively synthesize amplitude-only, phase-only, and complex-weighted phased arrays. The authors emphasize that PSO handles arbitrary nonlinear cost functions with simpler implementation and reduced bookkeeping compared to GA. Grimaccia et al. [31] compare Stud-GA, BPSO, and binary Social Network Optimization (bSNO) for thinning a 121-element antenna array. Similarly, Hybrid Taguchi Binary PSO (HTBPSO), introduced by [32], integrates the Taguchi method to improve local search and utilizes the "catfish effect" to avoid premature convergence. These studies firmly establish BPSO and its variants as well-founded, high-performing tools for discrete antenna synthesis and adaptive array design.

3. Materials and Methods

3.1. Particle Swarm Optimization

Each individual of the cluster, hereafter called a "particle", is initialized to a random position, within the search space, described by a d-dimensional vector x_n^0, where $1 \leq n \leq N$, N the total number of particles, and 0 refers to the initial time step. Each position x_n^i corresponds to a specific value of the fitness function, $f(x_n^i)$, where i is the time step or iteration number. In addition, each particle has a velocity vector v_n^i that changes dynamically at the i-th iteration. Finally, each particle has access to the best historical position, p_n^i, it has found itself, as well as the best position, g^i, found by the swarm as a whole. In fact, the best position refers to the position vector for which the fitness function takes its smallest (or largest, depending on the problem) value.

The most important aspect of the algorithm is how the velocity is updated at each iteration, since this determines the trajectory of the search toward the optimal solution. The velocity update equation shown below consists of three components, each with a clear physical interpretation [33]:

1. **Inertia:** This is the natural tendency of all moving bodies to maintain their current direction and resist any change to it. The parameter ω, which varies from 0.0 to numbers close to 1.0 but, theoretically, without any specific limit, determines how strongly a particle retains its previous velocity.

2. **Cognitive part:** It represents the effect exerted on the movement of the particle by its personal experience, and is reflected in the parameters c_1 and ϕ_1. The parameter c_1 is a fixed positive number determined at the beginning of the algorithm. A large value of c_1 leads to a wider search since each particle tends to move around its personal optimum. On the other hand, ϕ_1 is a random real-valued number between 0.0 and 1.0 that is updated at each iteration [34].
3. **Social influence:** Accordingly, the third term suggests the influence of g^i on the movement of each particle. Here, an increased value of c_2 leads to a convergence of the particles around the known best position and promotes local exploration of this region, while ϕ_2 again varies randomly between 0.0 and 1.0 at each iteration.

Combining all the above, we arrive at the following equation which describes the update of the d-th velocity component of the n-th particle from iteration i to $i+1$:

$$v_n^{i+1}[d] = \omega \times v_n^i[d] + c_1 \times \phi_1 \times (p_n^i[d] - x_n^i[d]) + c_2 \times \phi_2 \times (g^i[d] - x_n^i[d]) \quad (1)$$

with $1 \leq d \leq D$, where D is the dimension of the vectors.

Also, the position vector is formed as follows:

$$x_n^{i+1}[d] = x_n^i[d] + v_n^{i+1}[d] \quad (2)$$

A flowchart depicting the original PSO algorithm is given in Figure 1.

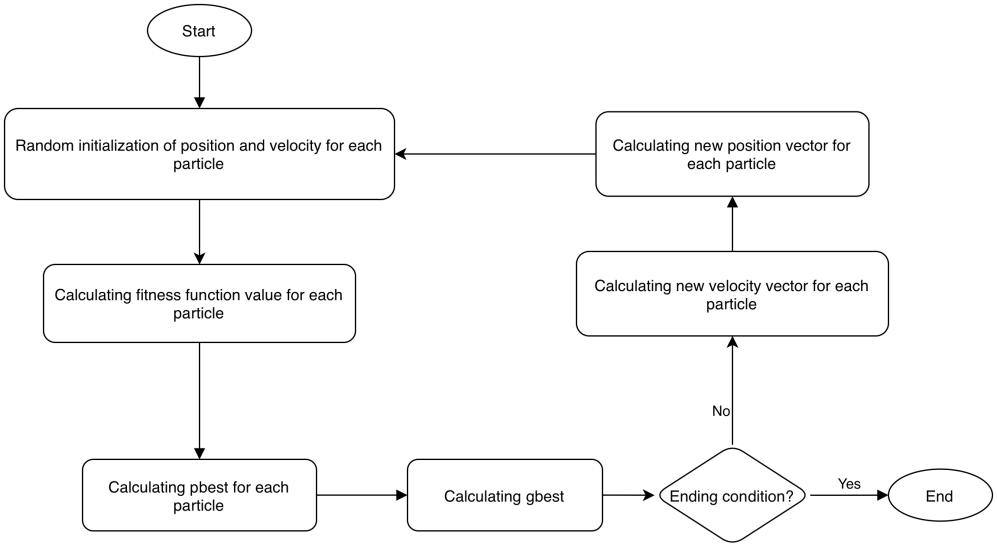


Figure 1. PSO algorithm flowchart.

3.2. PSO for Binary Space Optimization Problems

One of the most effective and accurate adaptations of PSO for binary space optimization problems is described in Afshinmanesh et al. [3]. Here, each particle is described by a bitstring—i.e., a sequence of binary digits (0 and 1)—that replaces the position coordinates. At the beginning of the algorithm, this bitstring is initialized randomly for each particle. The goal is again to find the bit sequence that yields the optimal value of the fitness function. In the movement of particles, the notion of velocity is used to capture the logic of the original PSO. Velocity is influenced by the three known factors: inertia, personal, and group best experience, while as one can see from the following equation, there is a one-to-one correspondence with the terms of (1). The fundamental difference lies in the fact

that, instead of the mathematical operations of multiplication, addition, and subtraction, logical operations at the bit level (bitwise operations) are used [33]. More specifically:

- Multiplication is replaced by logical AND ($\otimes$)
- Addition is replaced by logical OR ($\odot$)
- The subtraction is replaced by logical EXCLUSIVE OR ($\oplus$)

$$v_n^{i+1} = (\omega \otimes v_n^i) \odot \{c_1 \otimes (p_n^i \oplus x_n^i)\} \odot \{c_2 \otimes (g^i \oplus x_n^i)\} \tag{3}$$

Of course, this time, ω, c_1, and c_2 are not real-valued numbers but bitstrings containing as many binary digits as the position and velocity vectors, allowing logical operations to be performed smoothly. These bitstrings are randomly generated at each iteration according to a Bernoulli distribution. Consequently, at each iteration, three random bitstrings—ω, c_1, and c_2—are generated with the same length, but each is derived from a different Bernoulli distribution with respective probabilities Ω, C_1, and C_2. These factors are parameters of the algorithm and are initially defined by the user, taking real values strictly in the interval $[0.0, 1.0]$, since they represent probabilities. Having clarified the velocity update method, the position of each particle is easily calculated for the next iteration. Thus, the position bitstring for iteration $i + 1$ is obtained as follows:

$$x_n^{i+1} = x_n^i \oplus v_n^{i+1} \tag{4}$$

To introduce the maximum speed, V_{max}, the study of Afshinmanesh et al. [3] employs the process of negative selection, a mechanism observed in the immune system. Thus, V_{max} determines the extent of mutation that each velocity vector is allowed to undergo. The term mutation refers to the flipping of random bits in a particle's velocity bitstring from 1 to 0. The goal is to unexpectedly affect its next position and help it escape local minima, but, as explained below, it does not achieve it very effectively. The idea of mutation turns out to be extremely useful for optimizing complex functions, as the swarm often converges to local minima that are very close to the global optimum. Its implementation is presented in pseudocode form below:

```
C        #number of 1s in v
V_max    #max number of 1s allowed in v
while C > V_max:
    select a random 1 within v
    replace this 1 with 0
end
```

3.3. Boolean PSO with 0-Mutation

The technique described in the pseudocode above has been shown to be promising in several studies [3,33]. However, in this article, a different version is proposed that outperforms the previous ones. The innovation lies in the way the mutation process is performed. While in previous versions of BPSO, ones are identified in the velocity vector and flipped to zeros, here the reverse occurs: zeros are identified and flipped to ones, as shown in the pseudocode below:

```
C        #number of 0s in v
V_max    #max number of 0s allowed in v
while C > V_max:
    select a random 0 within v
    replace this 0 with 1
end
```

The idea for such a change arose from the need to introduce randomness into particle position update, so that they can escape local minima and perform a broader search. In older BPSO variants, the dominance of zeros in the velocity vector is more consistent with the notion of maximum velocity, but it also limits changes in the position vector. Under the XOR-based update rule, a velocity bit equal to zero leaves the corresponding position bit unchanged, while a velocity bit equal to one flips it. Therefore, a higher occurrence of ones increases exploration by producing more frequent changes in the position vector. More specifically, the reinterpretation of $V_{\max}$ from the maximum number of ones allowed in the velocity bitstring to the maximum number of zeros allowed alters the exploration mechanism of the algorithm. The key difference arises from the XOR-based position update. Under the XOR operator, a position bit flips if and only if the corresponding velocity bit equals 1. Therefore,

$$d_H(x_{i+1}, x_i) = w_H(v_{i+1}), \tag{5}$$

where $d_H(\cdot)$ denotes the Hamming distance and $w_H(\cdot)$ denotes the number of ones in the bitstring.

In the classical formulation, where $V_{\max}$ represents the maximum number of ones allowed in the velocity vector, we have

$$w_H(v_{i+1}) \leq V_{\max} \tag{6}$$

Consequently,

$$d_H(x_{i+1}, x_i) \leq V_{\max} \tag{7}$$

which directly restricts the particle's displacement in the search space. As the velocity progressively loses ones, it may converge to the zero vector, yielding

$$d_H(x_{i+1}, x_i) = 0, \tag{8}$$

and thus particle stagnation. Empirically, such behavior was observed in several cases in which the swarm became trapped in local minima during the first iterations.

In contrast, when $V_{\max}$ is redefined as the maximum number of zeros allowed in the velocity vector, we obtain

$$\#\{0 \text{ s in } v_{i+1}\} \leq V_{\max} \tag{9}$$

This implies

$$w_H(v_{i+1}) \geq L - V_{\max}, \tag{10}$$

where L is the bitstring length. Hence,

$$d_H(x_{i+1}, x_i) \geq L - V_{\max} \tag{11}$$

If $V_{\max} < L$ (that is true in our case because $V_{\max}$ is a percentage of L), then

$$d_H(x_{i+1}, x_i) > 0, \tag{12}$$

which guarantees a strictly positive displacement at every iteration and guarantees a nonzero bit-level displacement after mutation. This modification effectively imposes a lower bound on particle oscillation, forcing continued exploration of the search space and reducing the likelihood of premature convergence. As demonstrated in the results section, when the requirement for perturbation is gradually relaxed through adaptive scheduling, the algorithm achieves improved performance by first enforcing broad exploration and subsequently allowing controlled convergence to the maximum number of iterations.

During our research, evidence emerged to justify modifying the mutation process described in Afshinmanesh et al. [3] and orienting it towards velocity zeros rather than ones. As discussed above, substituting zeros for ones imposes a velocity constraint and thus yields more predictable paths, rather than introducing the randomness needed to untrap particles and encourage broader exploration. In contrast, substituting ones for zeros in the velocity vector effectively "redirects" particles in the desired way.

An empirical analysis of a 1000-iteration BPSO execution using "ones" mutation (i.e., replacing ones with zeros) demonstrates that the final global best position $g^{i_{max}}$ is identified prematurely, typically before the 100th iteration. This occurs because the continuous restriction of the velocity vector leads to abnormally fast convergence of the swarm. Consequently, the velocity tends toward zero, and particle stagnation is eventually encountered. In contrast, conducting the same process with "zeros" mutation (i.e., replacing zeros with ones) helps the swarm continue the search up to the last iteration of the algorithm.

To model the mutation, an additional factor is added alongside the existing parameters Ω, C_1, and C_2. This factor, called the mutation factor, takes real values from 0 to 1 and represents the maximum percentage of bits in the velocity bitstring that can be 0 in each iteration. Hence, V_{max} depends on the variable $v_mutation_factor$ and is derived as follows:

$$V_{max} = \lceil v_mutation_factor \cdot bitstring_length \rceil \tag{13}$$

4. Numerical Results

4.1. Numerical Benchmark Functions

In an attempt to demonstrate the effectiveness of the proposed method, several versions of the technique were tested on a wide collection of test functions. To determine the exact values of the parameters used in each case, a wide range of different parameter sets were applied to the Bukin N.6 function as given by Gavana [35]. This function was chosen because it is difficult to optimize reliably. Figure 2 shows the landscape of Bukin N.6. The formula of Bukin N.6 is given below:

$$\begin{aligned} f(x_1, x_2) &= 100\sqrt{|x_2 - 0.01x_1^2|} + 0.01|x_1 + 10|, \\ & x_1 \in [-15, -5], \quad x_2 \in [-3, 3]. \end{aligned} \tag{14}$$

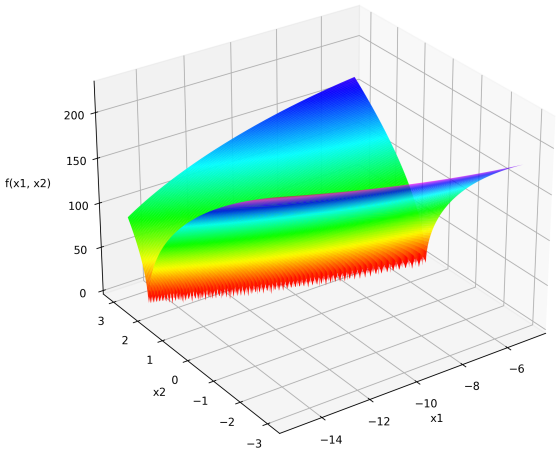


Figure 2. Bukin N.6 test function landscape from [35].

For example, in the case of a stable mutation factor and a stable omega factor, as described in Section 4.1.2, 900 possible parameter combinations are derived as follows:

$$\begin{cases} swarm_size = \{30\} \\ v_mutation_factor = \\ \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ inertia \quad factor \quad \Omega = \\ \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ cognitive \quad factor \quad C_1 = \{0.25, 0.50, 0.75\} \\ social \quad factor \quad C_2 = \{0.25, 0.50, 0.75\} \end{cases}$$

For each parameter combination, the algorithm was executed several times (e.g., 100 runs). After careful examination of the results, the optimal combination was selected based on two axes:

1. **Accuracy percentage:** The percentage of executions in which the algorithm successfully identified the global minimum of the Bukin N.6 function with significant accuracy (axis Y in Figure 3).
2. **Distance index:** The sum of the Euclidean distances between the actual global minimum and the final swarm best position $g^{i_{max}}$ across all executions that use the same parameter combination (axis X in Figure 3).

Diagrams such as Figure 3 were used to identify the most appropriate parameter combination. Clearly, the ideal set corresponds to the point lying highest and furthest to the left in the diagram.

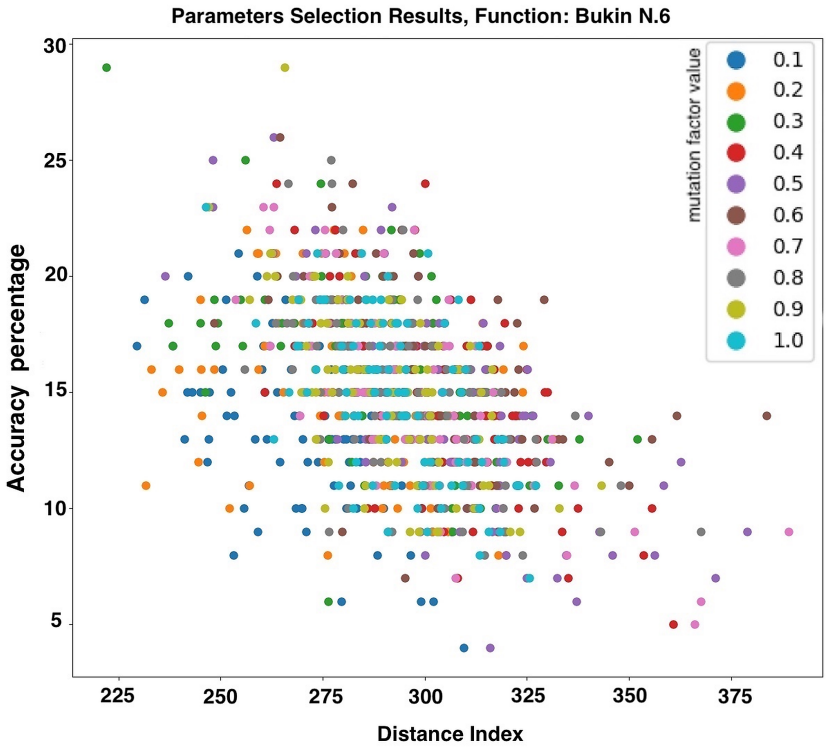


Figure 3. Parameter selection results for Bukin N.6 function with constant mutation and constant omega factor. Each point represents a different parameter combination, color-grouped by mutation factor value; 900 different parameter combinations were tested with 100 runs of the algorithm per combination.

Once the ideal parameter values for each variant are selected, the algorithm is applied to a set of 47 test functions to produce data on the accuracy of the particular version. The algorithm was executed 200 times for each fitness function, with 1000 iterations per execution. These functions, ranging from 1 to 30 dimensions, represent various landscape characteristics such as bowl-shaped, plate-shaped, and highly multimodal surfaces. A detailed list of these functions, including their designated identifiers and dimensionality used in this study, is provided in Table 1.

Table 1. Benchmark fitness functions: identifiers, names, real-valued dimensionality, and binary dimensionality.

#	Function Name	Dimensions (Number of Real-Valued Variables)	Number of Binary Variables
F1	Ackley ($a = 20, b = 0.2, c = 2\pi$)	30	690
F2	Beale	2	40
F3	Bohachevsky	2	50
F4	Booth	2	42
F5	Branin	2	40
F6	Bukin $N.6$	2	40
F7	Colville	4	84
F8	Cross-in-Tray	2	42
F9	De Jong $N.5$	2	48
F10	Dixon-Price	30	630
F11	Dropwave	2	40
F12	Easom	2	50
F13	Eggholder	2	54
F14	Forrester	1	17
F15	Goldstein-Price	2	38
F16	Gramacy & Lee Function	1	18
F17	Griewank	30	810
F18	Hartmann 3-D	3	51
F19	Hartmann 4-D	4	68
F20	Hartmann 6-D	6	102
F21	Holder Table	2	42
F22	Langermann	2	40
F23	Levy	30	630
F24	Levy $N.13$	2	42
F25	Matyas	2	42
F26	McCormick	2	40
F27	Michalewicz	30	570
F28	Perm $0, d, b$	30	690
F29	Perm d, b	30	690
F30	Powell	32	640

Table 1. *Cont.*

#	Function Name	Dimensions (Number of Real-Valued Variables)	Number of Binary Variables
F31	Power Sum Function	4	76
F32	Rastrigin	30	600
F33	Rosenbrock	30	630
F34	Rotated Hyper-Ellipsoid	30	720
F35	Schaffer *N*.2	2	50
F36	Schaffer *N*.4	2	50
F37	Schwefel	30	810
F38	Shekel	4	80
F39	Shubert	2	42
F40	Six-Hump Camel	2	39
F41	Sphere	30	600
F42	Styblinski–Tang	30	600
F43	Sum Squares Function	30	630
F44	Sum of Different Powers Function	30	540
F45	Three-Hump Camel	2	40
F46	Trid	30	840
F47	Zakharov	30	630

It should be clarified that, although most of the benchmark functions employed in this study are defined in continuous (real-valued) domains, the proposed BPSO operates in a binary search space. To bridge this gap, each real-valued decision variable is encoded independently into a fixed-length binary string using a standard binary-to-decimal mapping with a configurable precision. In this work, five decimal digits of precision are used for all variables. To establish the correspondence between real-valued and binary variables, we apply Equation (15) to determine the number of bits required for each real-valued variable, based on its range and the desired precision in number of decimal places (d). The total binary dimensionality of the search space is then obtained as the sum of the bits required for all variables, as expressed in Equation (16), and reported for each function in Table 1. Consequently, the binary optimization algorithms implemented in this work are evaluated on problems with tens or even hundreds of binary dimensions.

$$b_i = \left\lceil \log_2\left((\text{bound_max}_i - \text{bound_min}_i) \cdot 10^d \right) \right\rceil \tag{15}$$

$$\text{bit_dim} = \sum_{i=1}^{n} b_i \tag{16}$$

More specifically, the optimization state of each particle is maintained in binary form. The BPSO operators, namely velocity update, mutation, and XOR-based position update, are applied in the binary domain. The resulting bitstring is then decoded into its corresponding real-valued vector before evaluating the objective function. This encoding–decoding procedure is applied consistently across all benchmark functions considered in this study. This procedure enables the Boolean PSO framework to be consistently applied to continuous benchmark problems while maintaining controllable numerical precision. It is worth noting

that the encoding–decoding procedure introduces additional computational overhead compared to classical real-valued PSO, since each iteration requires binary transformation and bitwise operations. Consequently, the proposed approach is not intended to outperform real-valued PSO in raw computational speed for continuous problems. However, the objective of this study is not computational acceleration but the investigation of search dynamics under a Boolean framework. Moreover, for moderate dimensionalities and typical benchmark functions, the additional cost remains limited, as bitwise operations are computationally inexpensive and fitness evaluation often dominates the overall runtime. In practical terms, the proposed Boolean formulation is primarily advantageous in inherently discrete problems, such as antenna thinning, where no encoding overhead is required, and the binary representation is natural.

4.1.1. No Mutation and Constant Inertia Factor

The simplest form is an implementation of the algorithm without mutation, which is equivalent to using a mutation factor value of 1. As a result, the introduction of randomness in the configuration of the velocity vector and, by extension, the position vector is absent, making it difficult for particles to escape local minima. For this reason, it is observed that the success rate is quite low across many fitness functions (Figure 4). This version is denoted as Boolean PSO-Constant Inertia (BPSO-CI).

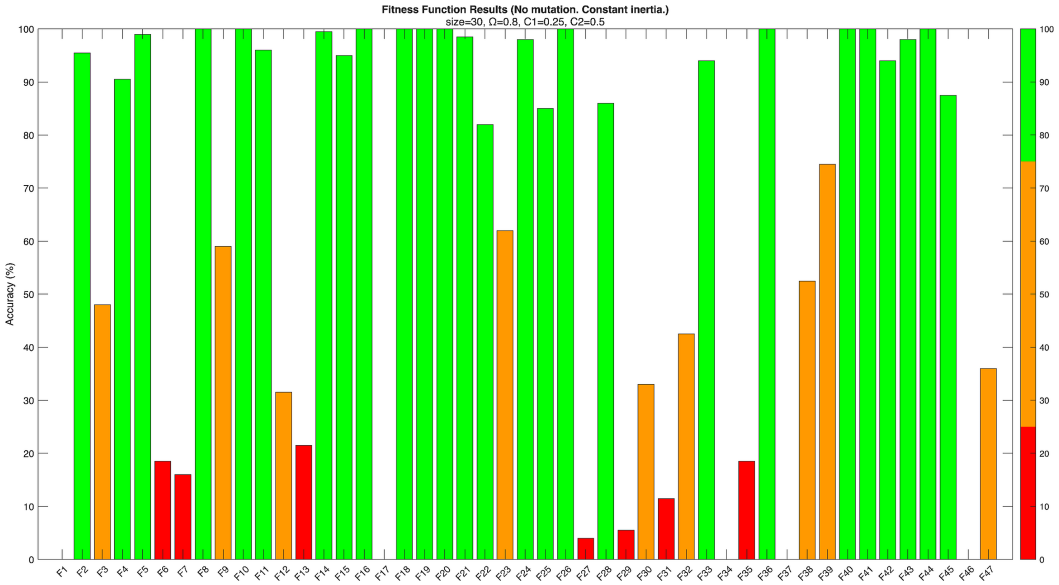


Figure 4. Accuracy of BPSO without mutation (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, \Omega = 0.8, C_1 = 0.25, C_2 = 0.50$.

4.1.2. Constant Mutation Factor and Constant Inertia Factor

In this case, fixed values are chosen for both the mutation factor and the inertia factor (Ω), requiring, however, the presence of mutation (thus $v_mutation_factor < 1$). This version is denoted as Boolean PSO–Constant Mutation–Constant Inertia (BPSO-CM-CI). As shown in Figure 5, there is a clear improvement in the success rate for a significant number of test cases, which have either entered into the green region or reached a 100% success rate. At this point, it is useful to clarify that in the accuracy figures, green color represents accuracy over 75%; orange color represents accuracy between 25% and 75%; and red color represents accuracy below 25%.

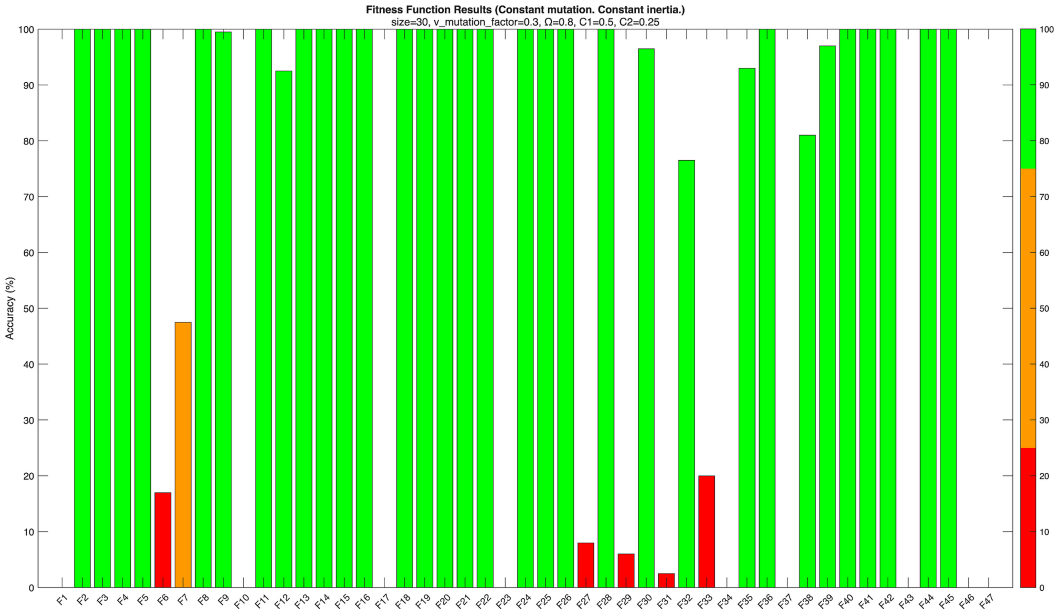


Figure 5. Accuracy of BPSO for constant mutation factor and constant inertia factor (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30$, $v_mutation_factor = 0.3, \Omega = 0.8, C_1 = 0.5, C_2 = 0.25$.

To further emphasize the importance of using mutation in the algorithm and of selecting an appropriate value for this parameter, the algorithm was also applied to the Powell function (Equation (17)), using different parameter combinations:

$$f(x) = \sum_{i=1}^{d/4} \begin{aligned}&\left[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2\right. \\ &\left.+(x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4\right]\end{aligned} \quad (17)$$

Each possible combination resulting from the following sets of values was tested:

$$\begin{cases} swarm_size = \{30\} \\ v_mutation_factor = \\ \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ inertia \quad factor \quad \Omega = \\ \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \\ cognitive \quad factor \quad C_1 = \{0.2, 0.4, 0.6, 0.8\} \\ social \quad factor \quad C_2 = \{0.2, 0.4, 0.6, 0.8\} \end{cases}$$

The results are shown in Figure 6 color-grouped by the value of the mutation factor. This function is considerably easier to optimize than Bukin N.6 as given in Gavana [35]. Nevertheless, it is observed that depending on the value of the mutation factor, the success rate shows large variations. Focusing on the light blue region, which corresponds to the absence of mutation (or equivalently $v_mutation_factor = 1.0$), it is evident that the performance of the algorithm falls clearly short compared to other options.

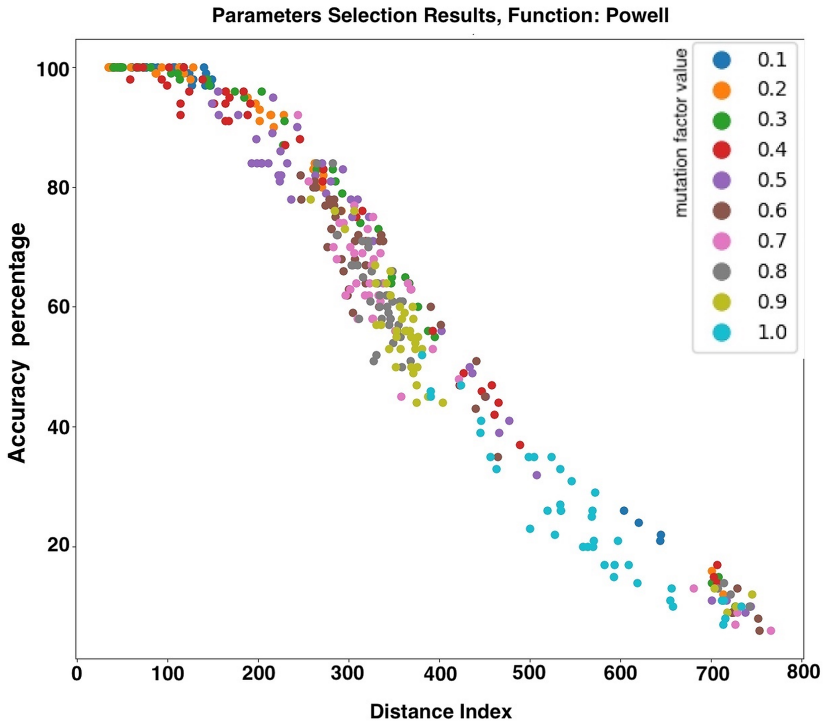


Figure 6. Efficiency of 400 different parameter sets for 100 runs of the algorithm with constant mutation and constant omega factor, color-grouped by mutation factor value.

4.1.3. Linear Change of Mutation Factor and Constant Inertia Factor

In this variant, the inertia factor, Ω, remains constant, while the mutation factor follows a linear variation from an initial to a final point over the course of the algorithm's iterations. The initial mutation factor is smaller than the final one so that the algorithm is more affected by random mutations at the beginning and is left free to converge during later stages. This version is denoted as Boolean PSO-Linear Mutation-Constant Inertia (BPSO-LM-CI). The results after running this technique for all fitness functions are shown in Figure 7. An additional improvement is observed, supporting the hypothesis that performance is enhanced by beginning with a more extensive and unconstrained search and then gradually reducing the degree of mutation to facilitate algorithm convergence.

4.1.4. Constant Mutation Factor and Linear Change of Inertia Factor

In the next attempt, the mutation factor is stabilized, and attention is turned to the inertia factor, Ω, for which linear variation is chosen during the algorithm iterations. As shown in the velocity update (3), increasing the inertia factor enhances the contribution of the previous velocity to the velocity at the next iteration. Introducing a linear reduction in inertia provides greater autonomy to particle motion in the early stages, while gradually allowing the other two terms of (3) to dominate in later iterations. Over time, the velocity vector is increasingly determined by g^i and p_n^i, thereby encouraging convergence and thorough local search. This version is denoted as Boolean PSO–Constant Mutation–Linear Inertia (BPSO-CM-LI). As shown in Figure 8, this BPSO variant produces significantly better results than variants with no mutation and fixed inertia or with fixed mutation and fixed inertia, but it falls short of the performance achieved by the BPSO variant that uses variable mutation and fixed inertia.

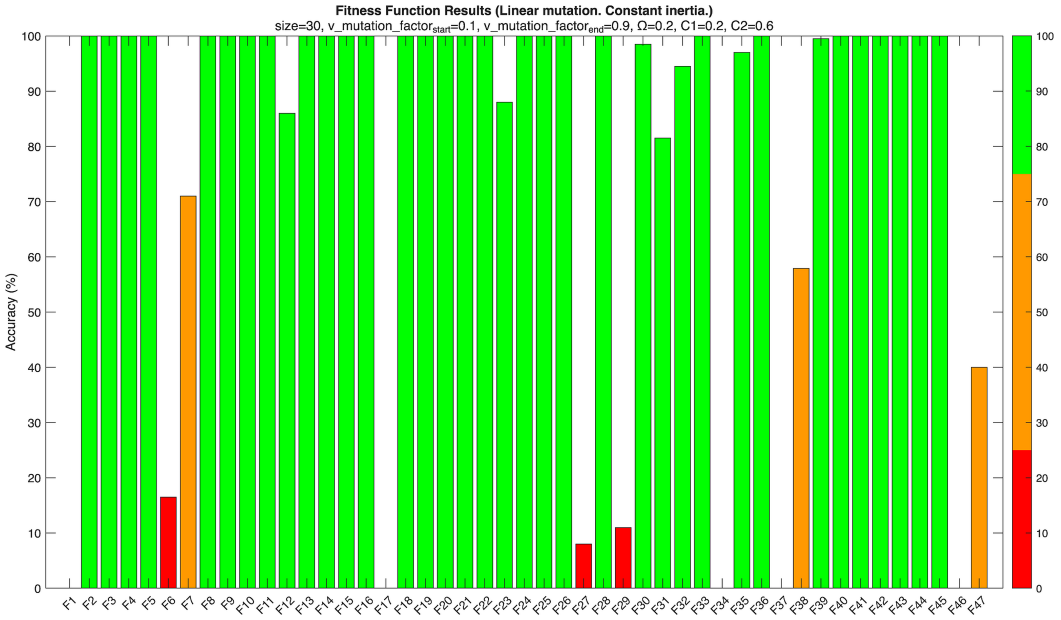


Figure 7. Accuracy of BPSO for the linear change of the mutation factor and constant inertia factor (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, initialv_mutation_factor = 0.1, finalv_mutation_factor = 0.9, \Omega = 0.2, C_1 = 0.2, C_2 = 0.6$.

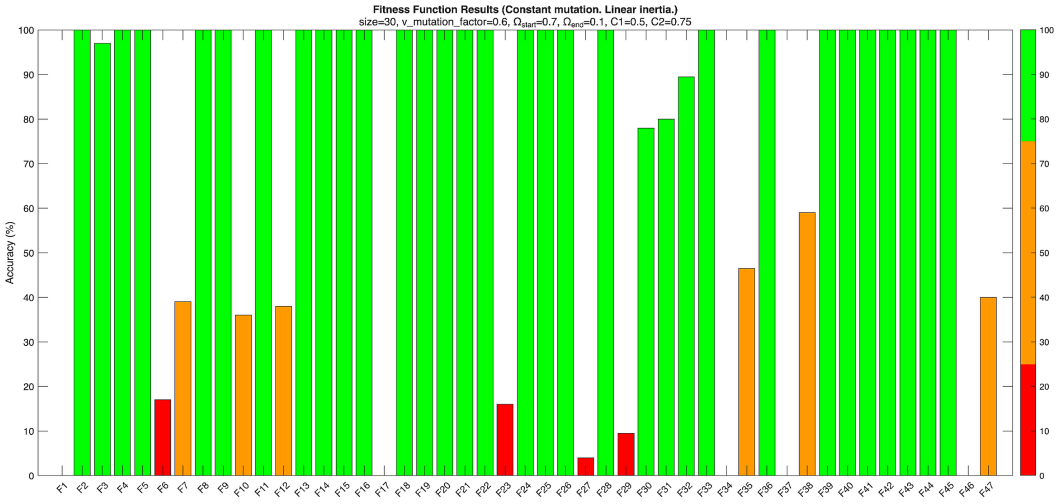


Figure 8. Accuracy of BPSO for a constant mutation factor and linear change of inertia factor (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, v_mutation_factor = 0.6, initial\ \Omega = 0.7, final\ \Omega = 0.1, C_1 = 0.5, C_2 = 0.75$.

4.1.5. Linear Change of Mutation Factor and Linear Change of Inertia Factor

In an attempt to identify the BPSO variant with optimal efficiency, a simultaneous linear variation was applied to both the mutation factor and the inertia factor. Since the independent application of each parameter variation resulted in considerable improvements, their combined effect was deemed worth investigating. This version is denoted as Boolean PSO-Linear Mutation-Linear Inertia (BPSO-LM-LI). As shown in Figure 9, this combination leads to the best-performing version of the algorithm. The majority of the fitness functions are now in the green region with 35 out of 47 functions achieving an over 75% success rate.

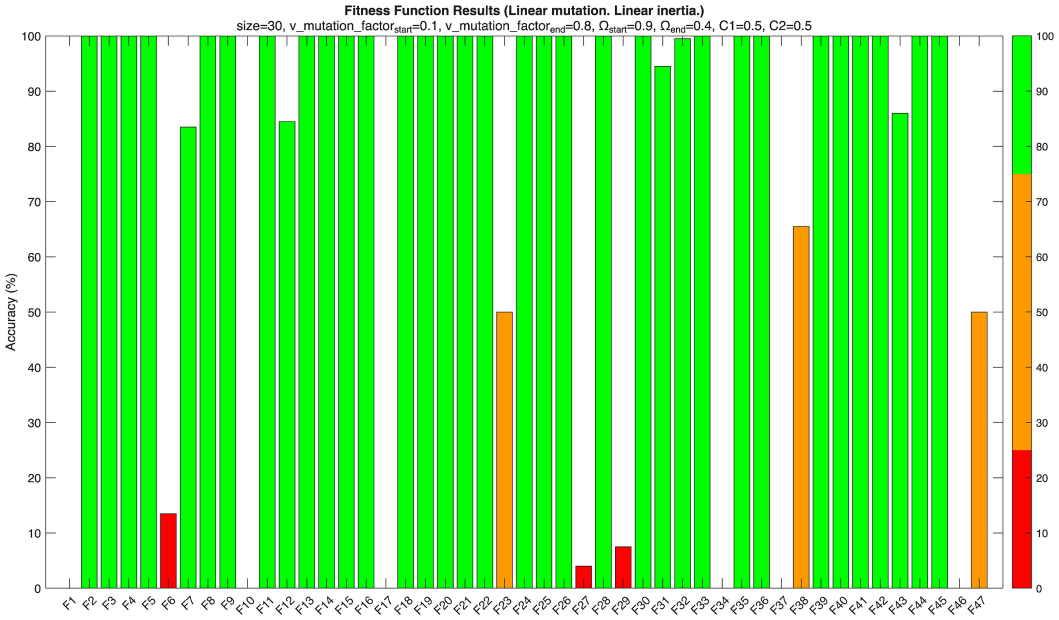


Figure 9. Accuracy of BPSO for linear change of mutation factor and linear change of inertia factor (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, initial\ v_mutation_factor = 0.1, final\ v_mutation_factor = 0.8, initial\ \Omega = 0.9, final\ \Omega = 0.4, C_1 = 0.5, C_2 = 0.5$.

4.2. Other Algorithms

To further validate the effectiveness of the proposed approach, we compare our results with other optimization algorithms. To keep the comparison fair, all algorithms were initialized with a population of 30, and the maximum number of iterations was set to 1000.

4.2.1. Binary PSO

As a starting point, we used the original Binary PSO (BinPSO) formulation of Kennedy and Eberhart [2], where velocity is interpreted as the probability of flipping a bit through a sigmoid transfer function (Equation (18)). In this classical framework, the real-valued velocity is mapped into the interval [0,1] using the sigmoid function, and the position update is performed according to a probabilistic rule described in Equation (19).

$$S\left(v_n^{i+1}[d]\right) = \frac{1}{1 + e^{-v_n^{i+1}[d]}} \tag{18}$$

$$x_n^{i+1}[d] = \begin{cases} 1, & \text{if rand}() < S\left(v_n^{i+1}[d]\right) \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

The results are shown in Figure 10. Even by visual inspection, it is easy to see that the proposed method achieves superior results for the majority of the functions.

4.2.2. Binary Grey Wolf Optimization Algorithm

Next, based on the study of Emary et al. [36] we implement the Binary Grey Wolf Optimization Algorithm (BGWOA). BGWOA is the binary version of the Grey Wolf Optimization Algorithm, which is a bio-inspired algorithm that mimics the hunting procedure of grey wolf packs [37]. Here, we used the bGWO2 version of [36] together with the proposed parameter values. Accuracy results are presented in Figure 11.

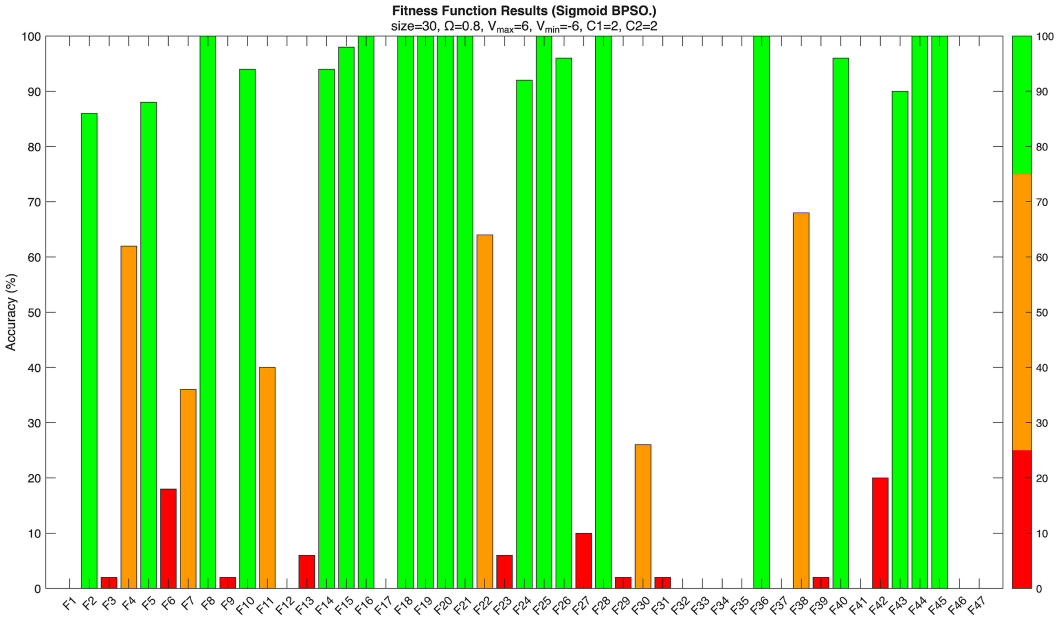


Figure 10. Accuracy of BinPSO (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, \Omega = 0.8, V_{max} = 6, V_{min} = -6, C_1 = 2, C_2 = 2$.

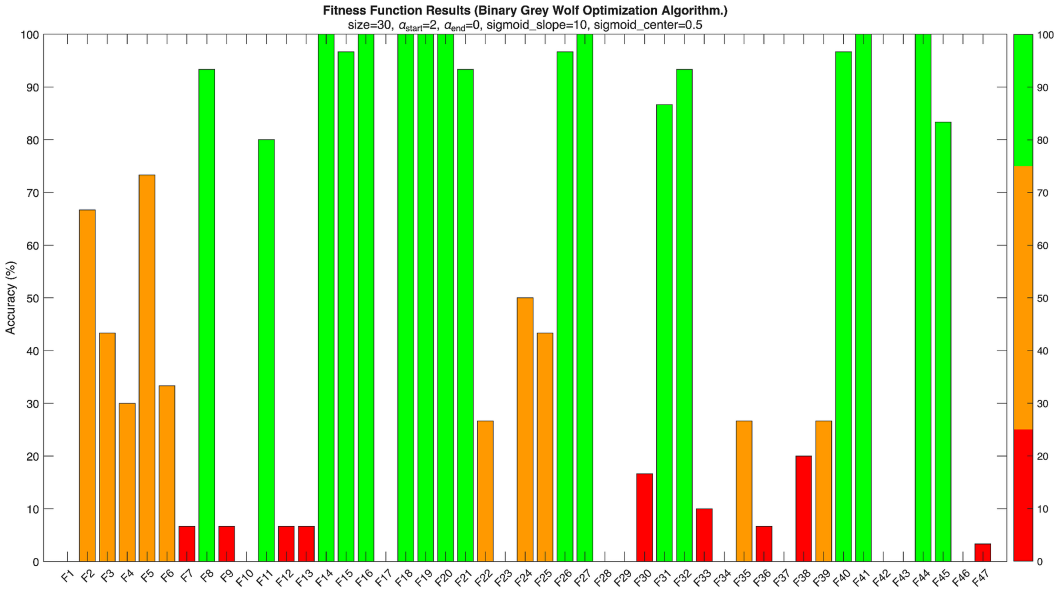


Figure 11. Accuracy of the Binary Grey Wolf Optimization Algorithm (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, initial\ \alpha = 2, final\ \alpha = 0, sigmoid_slope = 10, sigmoid_center = 0.5$.

4.2.3. Binary Genetic Algorithm

To continue, we use a Binary Genetic Algorithm (BGA), where each candidate solution is represented as a chromosome composed of binary digits. At every iteration, parent chromosomes are chosen from the population using a selection rule, recombined through crossover to generate offspring, and then randomly altered by bit-wise mutation [38]. The version used here is described in [39]. From Figure 12, we observe that BGA achieves higher accuracy than BinPSO and BGWOA but lower accuracy than the proposed method.

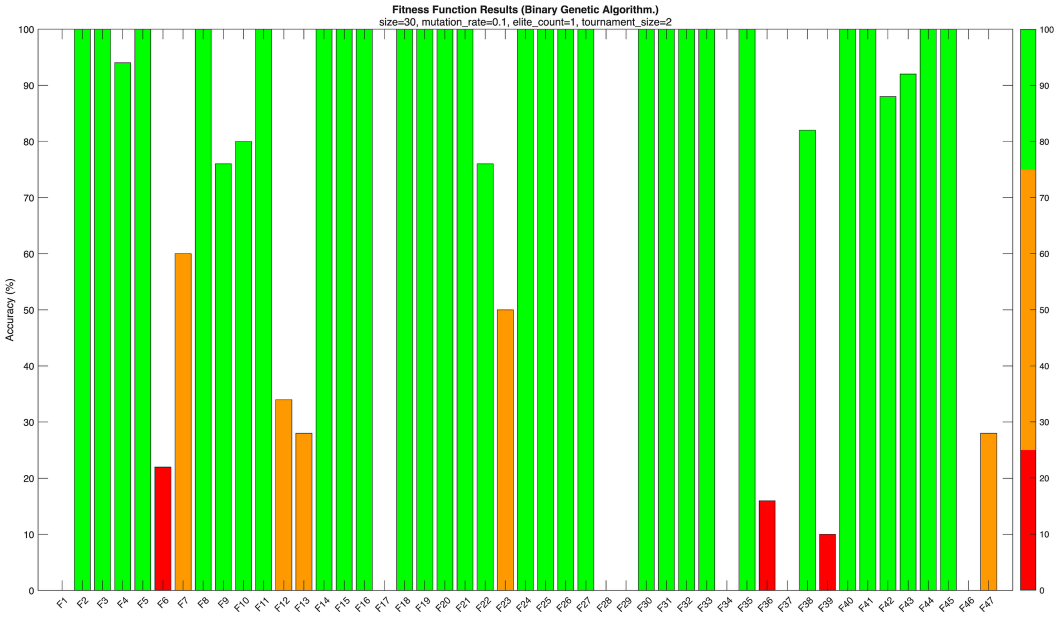


Figure 12. Accuracy of the Binary Genetic Algorithm (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, mutation_rate = 0.1$, $elite_count = 1, tournament_size = 2$.

4.2.4. Binary Differential Evolution Algorithm

Finally, we implement a binary version of the classical Differential Evolution (DE) algorithm, as described in [40]. The Binary Differential Evolution Algorithm (BDE) is a population-based optimization method that, like the proposed method, relies on the principles of mutation and crossover to explore the search space. From Figure 13, we observe that BDE achieves higher accuracy than BinPSO, BGWOA, and BGA but not as high as the proposed BPSO-LM-LI variant.



Figure 13. Accuracy of the Binary Differential Evolution Algorithm (percentage of finding the global minimum for each fitness function after 200 runs). Parameter values: $swarm_size = 30, cr = 0.3$, $F = 1, b = 6$.

4.3. Summary Results

In this subsection, a more detailed statistical analysis of the results is presented. The statistical comparison is based on the Friedman test, the Wilcoxon signed-rank test, and rank counting. All of them are well-known tools for this kind of analysis [41]. The Friedman test is a nonparametric test that compares the mean ranks of methods across all benchmark

functions, while the Wilcoxon signed-rank test assesses whether pairwise differences between methods are statistically significant [41]. Rank counting involves counting the number of times each algorithm ranks first, second, etc., across all benchmark functions. For the rank-count table, note that if two or more algorithms achieve the same performance on a given function, they are assigned the same rank for that function, and the next rank(s) are skipped accordingly. For example, if two algorithms tie for first place, both are assigned a rank of 1, and the next algorithm is assigned a rank of 3 (skipping rank 2).

Table 2 reports the mean values obtained by all versions of the proposed algorithm, while Table 3 reports the mean values obtained by BPSO-LM-LI and the algorithms described in Sections 4.2.1–4.2.4, with all mean values computed over 200 independent trials. One may notice that the BPSO-LM-LI variant obtains the lowest values in most cases. This is also confirmed by the Friedman ranking shown in Table 4. Moreover, according to Table 5, BPSO-LM-LI ranks first in 31 out of 47 test functions. The results of the Wilcoxon signed-rank test are reported in Table 6. It is clear that BPSO-LM-LI outperforms all other algorithms except BPSO-LM-CI. The two algorithms perform almost identically. The *p*-values show that BPSO-LM-LI is significantly better than all algorithms except BPSO-LM-CI.

Results indicate that adding a mutation clearly improves the performance of the algorithm. Moreover, adapting the mutation linearly during iterations leads to further improvement. Adapting the inertia factor also offers improvement, but not of the same magnitude. Finally, using linear adaptation of both parameters leads to the best version, BPSO-LM-LI, which is better than BPSO-LM-CI, but the gain is marginal rather than decisive.

According to the Friedman test shown in Table 7, BPSO-LM-LI achieves a better average rank than all the other methods. In fact, according to Table 8, BPSO-LM-LI ranks first in 27 functions and second in 10 functions, establishing it as first or runner-up in 37 functions out of the total of 47. Finally, the Wilcoxon signed-rank test results are reported in Table 9, showing that, according to the *p*-values, BPSO-LM-LI is significantly better than all other algorithms, except BDE. BDE is a strong competitor, and the statistical difference between BPSO-LM-LI and BDE is less significant, with a *p*-value of 0.335. However, the first place for BPSO-LM-LI in the other two tests, namely the Friedman test and the rank-count analysis, suggests an advantage of BPSO-LM-LI over BDE, although the performance of the algorithms remains problem-dependent.

Table 2. Mean values for 47 benchmark functions using the dimensionality listed in Table 1. Bold font indicates the lower values. Comparison among the proposed BPSO variants.

Function	BPSO-CI	BPSO-CM-CI	BPSO-LM-CI	BPSO-CM-LI	BPSO-LM-LI
F1	1.66×10^{1}	1.66×10^{1}	1.49×10^{1}	1.72×10^{1}	$\mathbf{1.46 \times 10^{1}}$
F2	6.23×10^{-2}	2.00×10^{-4}	2.00×10^{-4}	4.40×10^{-3}	$\mathbf{0.00 \times 10^{0}}$
F3	2.05×10^{1}	2.75×10^{-1}	4.73×10^{-2}	1.01×10^{0}	$\mathbf{3.79 \times 10^{-2}}$
F4	5.88×10^{-1}	1.00×10^{-3}	2.00×10^{-3}	9.60×10^{-3}	$\mathbf{7.00 \times 10^{-4}}$
F5	5.15×10^{-1}	$\mathbf{3.98 \times 10^{-1}}$	$\mathbf{3.98 \times 10^{-1}}$	4.49×10^{-1}	$\mathbf{3.98 \times 10^{-1}}$
F6	4.42×10^{-1}	7.02×10^{-1}	$\mathbf{1.66 \times 10^{-1}}$	1.78×10^{-1}	1.83×10^{-1}
F7	1.11×10^{2}	1.40×10^{1}	2.84×10^{0}	1.10×10^{1}	$\mathbf{2.84 \times 10^{0}}$
F8	$\mathbf{-2.06 \times 10^{0}}$	$\mathbf{-2.06 \times 10^{0}}$	$\mathbf{-2.06 \times 10^{0}}$	$\mathbf{-2.06 \times 10^{0}}$	$\mathbf{-2.06 \times 10^{0}}$
F9	3.30×10^{0}	1.01×10^{0}	$\mathbf{9.98 \times 10^{-1}}$	$\mathbf{9.98 \times 10^{-1}}$	$\mathbf{9.98 \times 10^{-1}}$
F10	3.88×10^{4}	5.16×10^{5}	$\mathbf{1.92 \times 10^{4}}$	1.25×10^{5}	7.23×10^{4}

Table 2. *Cont.*

Function	BPSO-CI	BPSO-CM-CI	BPSO-LM-CI	BPSO-CM-LI	BPSO-LM-LI
F11	-9.30×10^{-1}	-9.94×10^{-1}	-9.86×10^{-1}	-9.74×10^{-1}	-9.91×10^{-1}
F12	-3.09×10^{-1}	-8.60×10^{-1}	-8.56×10^{-1}	-3.78×10^{-1}	-8.43×10^{-1}
F13	-9.54×10^{2}	-9.60×10^{2}	-9.60×10^{2}	-9.60×10^{2}	-9.60×10^{2}
F14	-6.01×10^{0}	-6.02×10^{0}	-6.02×10^{0}	-6.02×10^{0}	-6.02×10^{0}
F15	1.24×10^{1}	3.01×10^{0}	3.00×10^{0}	3.01×10^{0}	3.00×10^{0}
F16	-8.68×10^{-1}	-8.69×10^{-1}	-8.69×10^{-1}	-8.69×10^{-1}	-8.69×10^{-1}
F17	7.79×10^{1}	3.79×10^{2}	5.24×10^{1}	1.47×10^{2}	7.05×10^{1}
F18	-3.82×10^{0}	-3.86×10^{0}	-3.86×10^{0}	-3.86×10^{0}	-3.86×10^{0}
F19	-3.00×10^{0}	-3.10×10^{0}	-3.12×10^{0}	-3.03×10^{0}	-3.13×10^{0}
F20	-3.12×10^{0}	-3.07×10^{0}	-3.20×10^{0}	-3.24×10^{0}	-3.20×10^{0}
F21	-1.91×10^{1}	-1.92×10^{1}	-1.92×10^{1}	-1.92×10^{1}	-1.92×10^{1}
F22	-3.72×10^{0}	-4.15×10^{0}	-4.14×10^{0}	-4.14×10^{0}	-4.15×10^{0}
F23	2.16×10^{1}	1.05×10^{2}	2.21×10^{1}	5.02×10^{1}	3.48×10^{1}
F24	1.26×10^{-1}	7.20×10^{-3}	0.00×10^{0}	0.00×10^{0}	0.00×10^{0}
F25	3.39×10^{-2}	4.00×10^{-4}	2.00×10^{-4}	5.00×10^{-4}	3.00×10^{-4}
F26	-1.90×10^{0}	-1.91×10^{0}	-1.91×10^{0}	-1.91×10^{0}	-1.91×10^{0}
F27	-1.39×10^{1}	-8.46×10^{0}	-1.28×10^{1}	-1.12×10^{1}	-1.20×10^{1}
F28	6.92×10^{3}	4.90×10^{1}	1.80×10^{0}	2.96×10^{0}	6.61×10^{-1}
F29	7.44×10^{2}	6.09×10^{1}	5.79×10^{0}	6.25×10^{0}	3.54×10^{0}
F30	9.52×10^{1}	2.40×10^{1}	2.84×10^{0}	2.05×10^{1}	1.22×10^{0}
F31	4.20×10^{1}	1.75×10^{1}	4.04×10^{-1}	4.75×10^{-1}	1.20×10^{-1}
F32	8.18×10^{0}	7.97×10^{0}	2.31×10^{0}	3.21×10^{0}	6.75×10^{-1}
F33	6.34×10^{4}	6.55×10^{5}	2.07×10^{4}	2.07×10^{4}	3.43×10^{4}
F34	8.04×10^{4}	2.63×10^{5}	4.66×10^{4}	4.66×10^{4}	4.19×10^{4}
F35	2.19×10^{-2}	8.00×10^{-4}	7.00×10^{-4}	1.52×10^{-2}	0.00×10^{0}
F36	3.03×10^{-1}	2.93×10^{-1}	2.93×10^{-1}	2.98×10^{-1}	2.93×10^{-1}
F37	5.90×10^{3}	9.05×10^{3}	5.36×10^{3}	6.05×10^{3}	5.52×10^{3}
F38	-4.66×10^{0}	-4.32×10^{0}	-5.69×10^{0}	-5.96×10^{0}	-5.71×10^{0}
F39	-1.82×10^{2}	-1.87×10^{2}	-1.87×10^{2}	-1.87×10^{2}	-1.87×10^{2}
F40	-1.01×10^{0}	-1.03×10^{0}	-1.03×10^{0}	-1.03×10^{0}	-1.03×10^{0}
F41	8.26×10^{-1}	3.40×10^{-1}	3.13×10^{-2}	2.53×10^{-2}	8.50×10^{-3}
F42	-1.91×10^{2}	-1.92×10^{2}	-1.96×10^{2}	-1.96×10^{2}	-1.96×10^{2}
F43	7.61×10^{2}	4.69×10^{3}	5.73×10^{2}	5.73×10^{2}	1.04×10^{3}
F44	1.30×10^{-1}	5.20×10^{-1}	2.00×10^{-2}	7.00×10^{-2}	3.00×10^{-2}
F45	4.40×10^{-2}	4.00×10^{-4}	1.00×10^{-4}	2.00×10^{-4}	0.00×10^{0}
F46	7.61×10^{5}	1.85×10^{6}	2.81×10^{5}	6.02×10^{5}	3.72×10^{5}
F47	2.61×10^{2}	4.23×10^{3}	2.38×10^{2}	2.28×10^{2}	2.34×10^{2}

Table 3. Mean values for 47 benchmark functions using the dimensionality listed in Table 1. Bold font indicates the lower values. Comparison among BPSO-LM-LI and other algorithms.

Function	BPSO-LM-LI	BinPSO	BGWOA	BGA	BDE
F1	1.46×10^{1}	2.08×10^{1}	1.93×10^{1}	1.37×10^{1}	1.77×10^{1}
F2	0.00×10^{0}	5.00×10^{-2}	4.34×10^{-1}	1.66×10^{-4}	3.01×10^{-5}
F3	3.79×10^{-2}	5.45×10^{2}	1.51×10^{2}	8.48×10^{-4}	4.25×10^{-10}
F4	7.00×10^{-4}	1.71×10^{0}	3.61×10^{0}	1.08×10^{-1}	2.32×10^{-4}
F5	3.98×10^{-1}	7.60×10^{-1}	6.58×10^{-1}	3.98×10^{-1}	3.98×10^{-1}
F6	1.83×10^{-1}	1.22×10^{1}	4.08×10^{0}	1.13×10^{-1}	8.69×10^{-2}
F7	1.81×10^{0}	9.24×10^{1}	4.22×10^{2}	4.85×10^{0}	1.15×10^{0}
F8	-2.06×10^{0}	-2.06×10^{0}	-2.04×10^{0}	-2.06×10^{0}	-2.06×10^{0}
F9	9.98×10^{-1}	5.22×10^{0}	1.08×10^{1}	1.18×10^{0}	9.98×10^{-1}
F10	7.23×10^{4}	1.55×10^{4}	8.30×10^{5}	6.30×10^{4}	1.32×10^{5}
F11	-9.91×10^{-1}	-5.90×10^{-1}	-8.96×10^{-1}	-9.90×10^{-1}	-1.00×10^{0}
F12	-8.43×10^{-1}	0.00×10^{0}	-5.34×10^{-2}	-3.40×10^{-1}	-1.00×10^{0}
F13	-9.60×10^{2}	-9.19×10^{2}	-8.48×10^{2}	-9.25×10^{2}	-9.59×10^{2}
F14	-6.02×10^{0}	-5.92×10^{0}	-6.01×10^{0}	-6.01×10^{0}	-6.02×10^{0}
F15	3.00×10^{0}	2.23×10^{1}	2.04×10^{1}	3.00×10^{0}	3.00×10^{0}
F16	-8.69×10^{-1}	-7.70×10^{-1}	-8.29×10^{-1}	-8.69×10^{-1}	-8.69×10^{-1}
F17	7.05×10^{1}	2.47×10^{2}	3.56×10^{2}	8.45×10^{1}	1.52×10^{2}
F18	-3.86×10^{0}	-3.51×10^{0}	-3.68×10^{0}	-3.86×10^{0}	-3.86×10^{0}
F19	-3.13×10^{0}	-2.53×10^{0}	-2.77×10^{0}	-3.12×10^{0}	-3.13×10^{0}
F20	-3.20×10^{0}	-1.94×10^{0}	-2.47×10^{0}	-3.27×10^{0}	-3.32×10^{0}
F21	-1.92×10^{1}	-1.91×10^{1}	-1.82×10^{1}	-1.92×10^{1}	-1.92×10^{1}
F22	-4.15×10^{0}	-3.77×10^{0}	-3.09×10^{0}	-4.08×10^{0}	-4.15×10^{0}
F23	3.48×10^{1}	1.10×10^{2}	8.44×10^{1}	3.05×10^{1}	4.75×10^{1}
F24	0.00×10^{0}	9.10×10^{-1}	1.24×10^{0}	3.09×10^{-5}	1.93×10^{-10}
F25	3.00×10^{-4}	0.00×10^{0}	1.02×10^{-1}	7.13×10^{-7}	9.09×10^{-13}
F26	-1.91×10^{0}	-1.51×10^{0}	-1.79×10^{0}	-1.91×10^{0}	-1.91×10^{0}
F27	-1.20×10^{1}	-5.87×10^{0}	-1.24×10^{1}	-1.61×10^{1}	-1.30×10^{1}
F28	6.61×10^{-1}	1.84×10^{2}	7.58×10^{87}	1.79×10^{80}	1.40×10^{77}
F29	3.54×10^{0}	5.10×10^{4}	3.21×10^{83}	1.51×10^{82}	6.96×10^{81}
F30	1.22×10^{0}	3.52×10^{2}	7.98×10^{3}	1.55×10^{3}	1.95×10^{3}
F31	1.20×10^{-1}	1.19×10^{3}	4.49×10^{2}	2.14×10^{0}	1.55×10^{-2}
F32	6.75×10^{-1}	6.15×10^{1}	2.47×10^{2}	1.90×10^{2}	2.44×10^{2}
F33	3.43×10^{4}	4.87×10^{6}	8.86×10^{5}	4.24×10^{4}	2.63×10^{4}
F34	4.19×10^{4}	4.06×10^{5}	2.25×10^{5}	5.21×10^{4}	8.93×10^{4}
F35	0.00×10^{0}	3.20×10^{-1}	1.18×10^{-1}	3.87×10^{-13}	2.41×10^{-11}
F36	2.93×10^{-1}	4.20×10^{-1}	3.42×10^{-1}	2.93×10^{-1}	2.93×10^{-1}
F37	5.52×10^{3}	1.08×10^{4}	5.63×10^{3}	4.11×10^{3}	4.41×10^{3}

Table 3. *Cont.*

Function	BPSO-LM-LI	BinPSO	BGWOA	BGA	BDE
F38	-5.71×10^{0}	-2.91×10^{0}	-1.91×10^{0}	-5.91×10^{0}	-9.51×10^{0}
F39	-1.87×10^{2}	-1.63×10^{2}	-1.42×10^{2}	-1.87×10^{2}	-1.87×10^{2}
F40	-1.03×10^{0}	-6.30×10^{-1}	-9.16×10^{-1}	-1.03×10^{0}	-1.03×10^{0}
F41	8.50×10^{-3}	2.97×10^{1}	1.06×10^{2}	2.56×10^{1}	4.56×10^{1}
F42	-1.96×10^{2}	-1.56×10^{2}	-6.52×10^{2}	-9.82×10^{2}	-9.12×10^{2}
F43	1.04×10^{3}	6.14×10^{2}	4.64×10^{3}	1.07×10^{3}	1.90×10^{3}
F44	3.00×10^{-2}	2.82×10^{0}	4.62×10^{-1}	4.72×10^{-3}	1.21×10^{-2}
F45	0.00×10^{0}	2.00×10^{-2}	1.36×10^{-1}	1.91×10^{-5}	4.55×10^{-11}
F46	3.72×10^{5}	1.08×10^{6}	6.33×10^{5}	6.23×10^{5}	1.02×10^{6}
F47	2.34×10^{2}	6.11×10^{10}	7.15×10^{7}	2.47×10^{2}	2.04×10^{2}

Table 4. Average rankings obtained by the Friedman test for Table 2 over 47 benchmark functions. Bold font indicates the lower values.

Algorithm	Ranking	Normalized
BPSO-CI	4.39	2.26
BPSO-CM-CI	3.57	1.84
BPSO-LM-CI	2.10	1.08
BPSO-CM-LI	2.99	1.54
BPSO-LM-LI	**1.95**	**1.00**

Table 5. Comparative study of the algorithms ranking first or second in Table 2 over 47 benchmark functions. Bold font indicates the higher values.

Algorithm	Ranking First	Ranking Second
BPSO-CI	3	10
BPSO-CM-CI	14	6
BPSO-LM-CI	24	**18**
BPSO-CM-LI	17	7
BPSO-LM-LI	**31**	10

Table 6. Wilcoxon signed-rank test results for Table 2 with a 0.05 significance level. Bold font indicates statistically significant *p*-values.

Comparison	R^+	R^-	*p* Value
BPSO-LM-LI vs. BPSO-CI	941.0	140.0	$\mathbf{1.21 \times 10^{-5}}$
BPSO-LM-LI vs. BPSO-CM-CI	614.0	16.0	$\mathbf{9.71 \times 10^{-7}}$
BPSO-LM-LI vs. BPSO-LM-CI	254.0	242.0	9.06×10^{-1}
BPSO-LM-LI vs. BPSO-CM-LI	508.5	121.5	$\mathbf{1.53 \times 10^{-3}}$

Table 7. Average rankings obtained by the Friedman test for Table 3 over 47 benchmark functions. Bold font indicates the lower values.

Algorithm	Ranking	Normalized
BPSO-LM-LI	**1.97**	**1.00**
BDE	2.14	1.09
BGA	2.38	1.21
BinPSO	4.14	2.10
BGWOA	4.37	2.22

Table 8. Comparative study of the algorithms ranking first or second in Table 3 over 47 benchmark functions. Bold font indicates the higher values.

Algorithm	Ranking First	Ranking Second
BPSO-LM-LI	**27**	**10**
BDE	25	9
BGA	16	**10**
BinPSO	4	4
BGWOA	0	0

Table 9. Wilcoxon signed-rank test results for Table 3 with a 0.05 significance level. Bold font indicates statistically significant p-values.

Comparison	$\mathbf{R}^+$	$\mathbf{R}^-$	p Value
BPSO-LM-LI vs. BinPSO	1001.0	80.0	$\mathbf{4.87 \times 10^{-7}}$
BPSO-LM-LI vs. BGWOA	1076.0	52.0	$\mathbf{6.02 \times 10^{-8}}$
BPSO-LM-LI vs. BGA	487.0	216.0	$\mathbf{4.24 \times 10^{-2}}$
BPSO-LM-LI vs. BDE	334.5	226.5	3.35×10^{-1}

4.4. Planar Array Thinning

In order to demonstrate the applicability of the proposed approach to a real-world engineering problem, we have selected a demanding and high-complexity problem from the antenna design domain. Thus, it was deemed appropriate to use the Boolean version of PSO for realistic problems in the field of applied electromagnetics to demonstrate its ability to produce valid and functional results.

4.4.1. Problem Description

Consider a rectangular antenna array of dimensions $M \times N$ (i.e., M elements along the x-axis and N elements along the y-axis), with element spacings d_x, d_y along the respective axes, as shown in Figure 14. The array factor (AF) is given by the following expression:

$$\begin{aligned} AF(\theta,\phi) = \sum_{m=1}^{M}\sum_{n=1}^{N} I_{mn} exp\{j \\ [(m-1)(kd_x sin\theta cos\phi + b_x) + \\ (n-1)(kd_y sin\theta sin\phi + b_y)]\} \end{aligned} \tag{20}$$

where I_{mn} is the excitation amplitude of the element at position (m, n), k is the wavenumber ($k = 2\pi/\lambda$), b_x and b_y are, respectively, the progressive phase shifts along the x and y-

axes, and finally θ and ϕ are, respectively, the polar and azimuth angles in the spherical coordinate system.

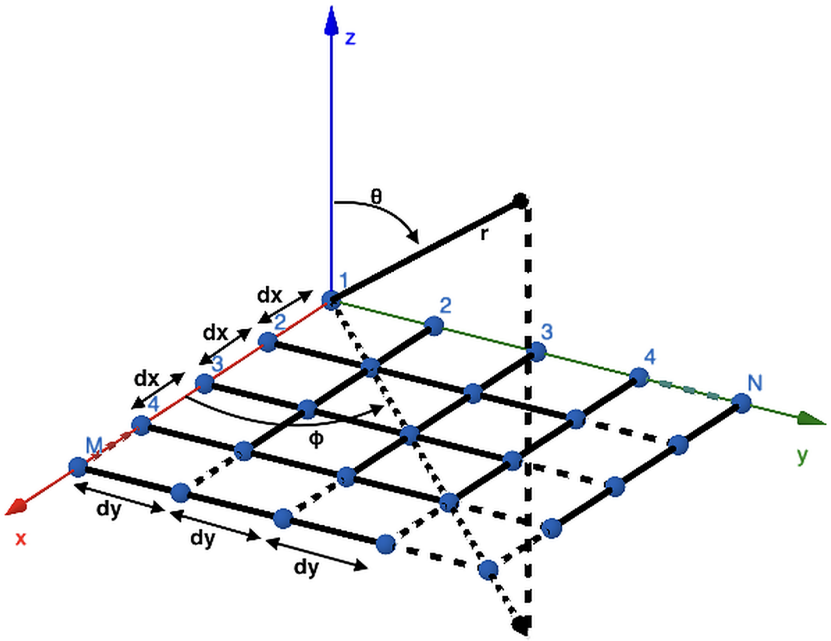


Figure 14. Planar antenna array.

According to Balanis [42], the directivity and the side lobe level (SLL) of the antenna array are respectively given by the following expressions:

$$D = \frac{4\pi|AF(\theta_0, \phi_0)|^2}{\int_0^{2\pi}\int_0^{\pi}|AF(\theta, \phi)|^2 sin\theta d\theta d\phi} \tag{21}$$

$$SLL = \frac{|AF(\theta_1, \phi_1)|}{|AF(\theta_0, \phi_0)|} \tag{22}$$

where (θ_0, ϕ_0) is the main lobe direction (global maximum of AF) and (θ_1, ϕ_1) is the direction of the greatest side lobe (local maximum with the highest AF value after the global maximum).

The main lobe direction (θ_0, ϕ_0) can be easily defined by adjusting suitable values for b_x and b_y according to the following expressions:

$$\begin{aligned} b_x &= -kd_x cos\phi_0 sin\theta_0 \\ b_y &= -kd_y sin\phi_0 sin\theta_0 \end{aligned} \tag{23}$$

In our problem, a rectangular 16×16 antenna array is employed ($M = N = 16$), with $d_x = d_y = \lambda/2$ (λ is the operating wavelength). The excitation amplitude is set to 1 for active elements or 0 for inactive elements. The goal is to find which elements should be activated ($I_{mn} = 1$) and which should be deactivated ($I_{mn} = 0$) in order to achieve the highest possible directivity and $SLL \leq -20$ dB.

It is evident that the problem, as defined above, is a discrete-variable problem, since the excitation amplitudes I_{mn} can take only two values, i.e., 1 or 0. Therefore, the requested solution is a sequence of binary digits, which means that the best position g^i derived by the proposed BPSO algorithm is a bitstring of $M \times N$ bits.

4.4.2. Fitness Function Construction

Creating an efficient fitness function that accurately captures the "suitability" of each solution is one of the most challenging aspects of optimization procedures and greatly facilitates the algorithm's convergence when designed properly. In this case, two design specifications must be met, and the weighted sum technique is used [43]. The first requirement concerns the maximization of directivity. Therefore, since the algorithm searches for the minimum, the first part of the fitness function can be:

$$\Phi(D) = -D(\text{dBi}) \tag{24}$$

The second requirement relates to maintaining the SLL close or below a certain threshold ($Threshold = -20$ dB). Any value below this threshold is considered equally appropriate. Accordingly, the second component of the fitness function is a penalty proportional to the distance of SLL from that threshold:

$$\Phi(SLL) = max[0, SLL(\text{dB}) - Threshold] \tag{25}$$

If the requirement is met ($SLL < -20$ dB), the penalty is zero. The fitness function will be expressed as a linear combination of the two terms given by (24) and (25). Because a design with slightly reduced directivity but compliant with the SLL condition would be more preferable, it is deemed appropriate to give more weight to the second term. Finally, the fitness function is obtained as:

$$\begin{aligned}\Phi(D, SLL) = &- 0.3D(\text{dBi}) + \\ &0.7max[0, SLL(\text{dB}) - Threshold]\end{aligned} \tag{26}$$

Note that the values of D and SLL in (26) are, respectively, in dBi and dB. The conversion is performed using the following expressions:

$$D(\text{dBi}) = 10 \times log_{10}(D) \tag{27}$$

$$SLL(\text{dB}) = 20 \times log_{10}(SLL) \tag{28}$$

The proposed BPSO variant, which achieved optimal results in previous tests, was used to evaluate the fitness function. This variant includes linear variation of both the mutation factor and the inertia factor over 1000 iterations, with parameter values:

$$\begin{cases} swarm_size = 30 \\ initial \quad v_mutation_factor = 0.1 \\ final \quad v_mutation_factor = 0.8 \\ initial \quad \Omega = 0.9 \\ final \quad \Omega = 0.4 \\ C_1 = 0.5 \\ C_2 = 0.5 \end{cases}$$

4.4.3. Design Case 1: Main Lobe Toward $\theta = 0^\circ$ and $\phi = 0^\circ$.

In the first case, the goal is to construct an antenna array with its main lobe directed toward the z-axis. Thus, by substituting $(\theta_0, \phi_0) = (0^\circ, 0^\circ)$ into (23), we obtain:

$$b_x = 0^\circ \qquad b_y = 0^\circ \tag{29}$$

The proposed BPSO variant was executed 25 times. The directivity and *SLL* values obtained from each run are presented in Figure 15. It is evident that the best design is the one with directivity 23.55 dBi and $SLL = -19.79$ dB. The appropriate design, referred to as Design 1 hereafter, is described by the following sequence of binary digits:

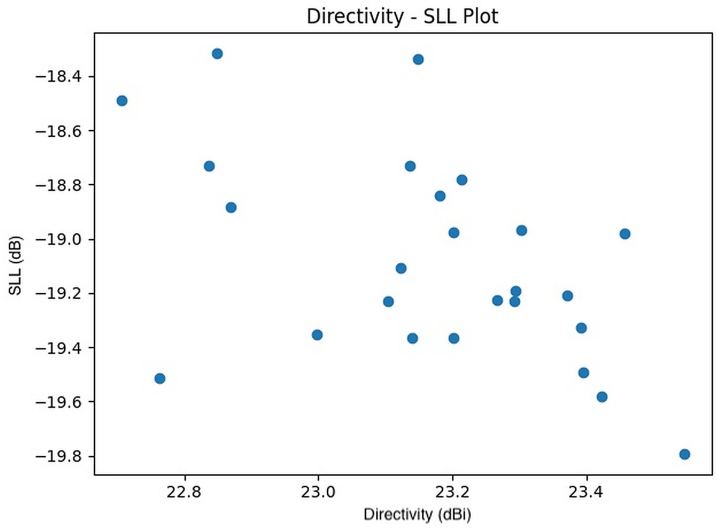


Figure 15. Directivity and *SLL* of the antenna array designs obtained from 25 runs of the algorithm (main lobe direction: $\theta_0 = 0^\circ, \phi_0 = 0^\circ$).

```
0011101100100100000011011101 1
0000111110100000011010110111 0001
1001101111011010111101111111 0001
1111011111111111000111111111 0011
1101111101111111101111111011 1101
1110111110111011101101111111 0110
0101111110100101110011110101 1000
0000000110100100001100111110 1101
```

Interestingly, only 164 of the 256 elements remain active, offering a significant reduction in energy consumption while still achieving strong performance. The structure of this antenna array is depicted in Figure 16, while Figures 17 and 18 show the corresponding radiation patterns.

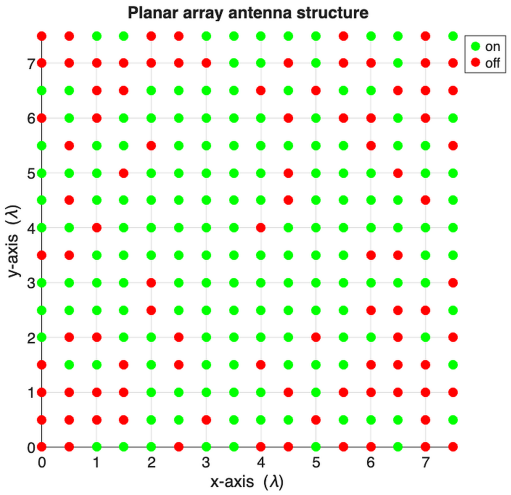


Figure 16. Graphic representation of active and inactive elements in the optimized Design 1 (164 active elements out of a total of 256).

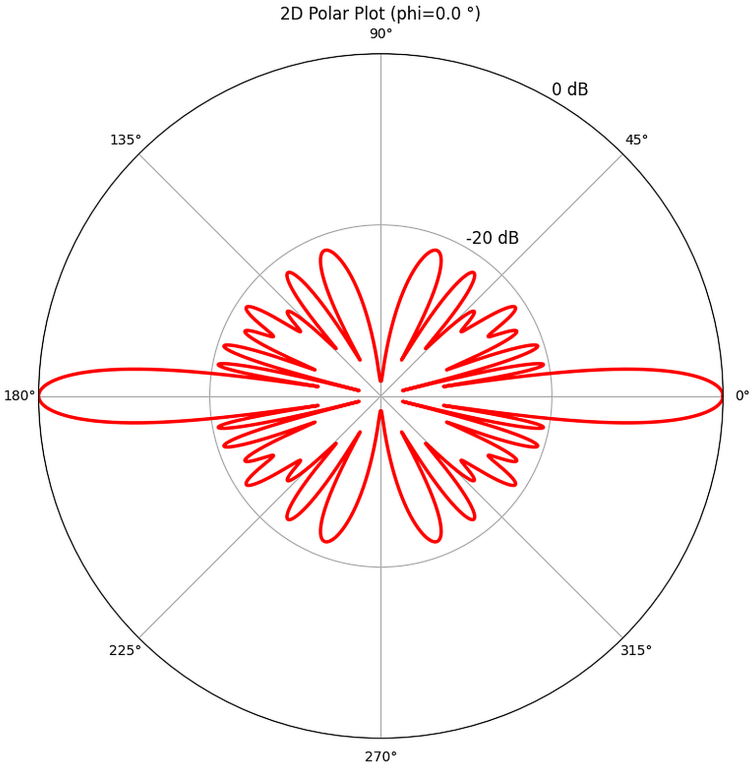


Figure 17. 2D radiation pattern of Design 1 with $\phi = 0^\circ$.

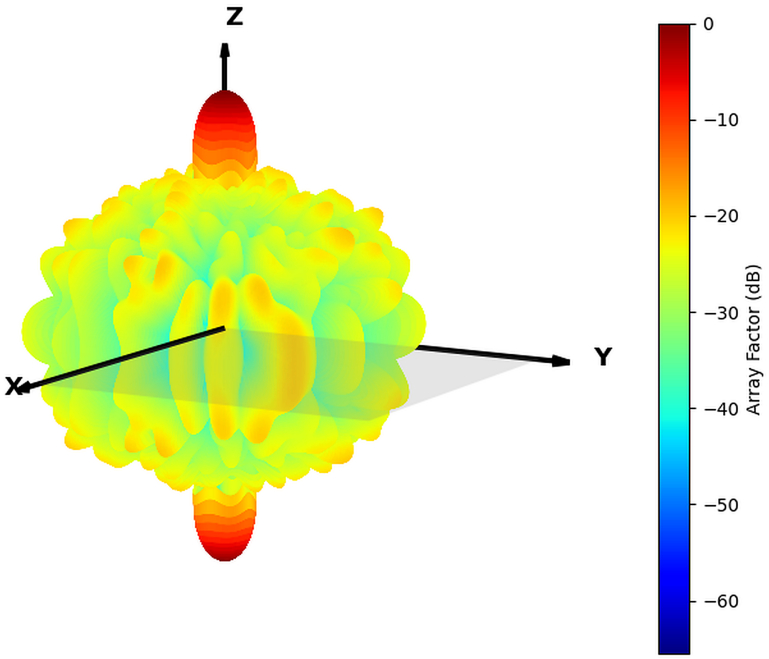


Figure 18. View of the 3D radiation pattern of Design 1.

4.4.4. Design Case 2: Main Lobe Toward $\theta_0 = 60^\circ$ and $\phi_0 = 30^\circ$.

In the second case, the goal is to construct an antenna array with its main lobe directed toward $(\theta_0, \phi_0) = (60^\circ, 30^\circ)$. From (23), we get

$$b_x = -\frac{3\pi}{4}\text{rad} = -135^\circ \qquad b_y = -\frac{\sqrt{3}\pi}{4}\text{rad} = -77.94^\circ \tag{30}$$

The proposed BPSO variant was executed 25 times. The directivity and *SLL* values obtained from each run are presented in Figure 19. It is evident that the best design is the one with directivity 21.19 dBi and $SLL = -19.69$ dB. The appropriate design, referred to as Design 2 hereafter, is described by the following sequence of binary digits:

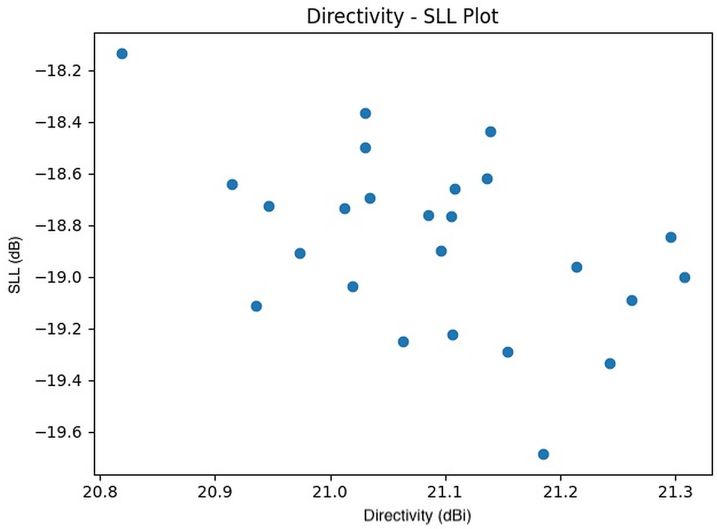


Figure 19. Directivity and *SLL* of the antenna array designs obtained from 25 runs of the algorithm (main lobe direction: $\theta_0 = 60^\circ, \phi_0 = 30^\circ$).

```
1000011110011000001011110110001
1000001110110010001111110110101 1
0100011111010100010011111011101
1111111101101001111110111111110
0011111111110011001111111111101
1011101111111101101011111111011
0101001101110100001010111000 1001
0101000101101011000101011101 1000
```

Interestingly, only 160 of the 256 elements remain active, offering a significant reduction in energy consumption while still achieving strong performance. The structure of this elemental array is depicted in Figure 20, while Figures 21–23 show the corresponding radiation patterns.

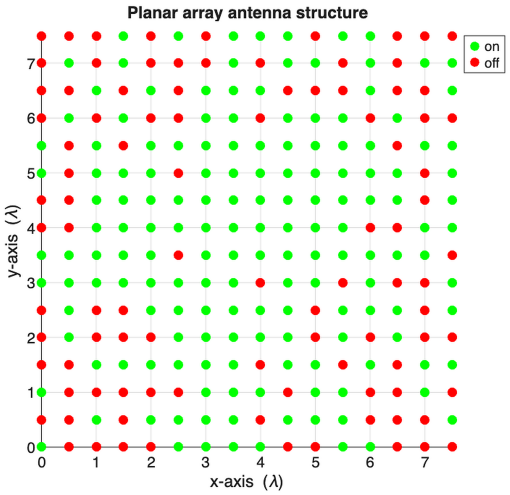


Figure 20. Graphic representation of active and inactive elements in optimized Design 2 (160 active elements out of a total of 256).

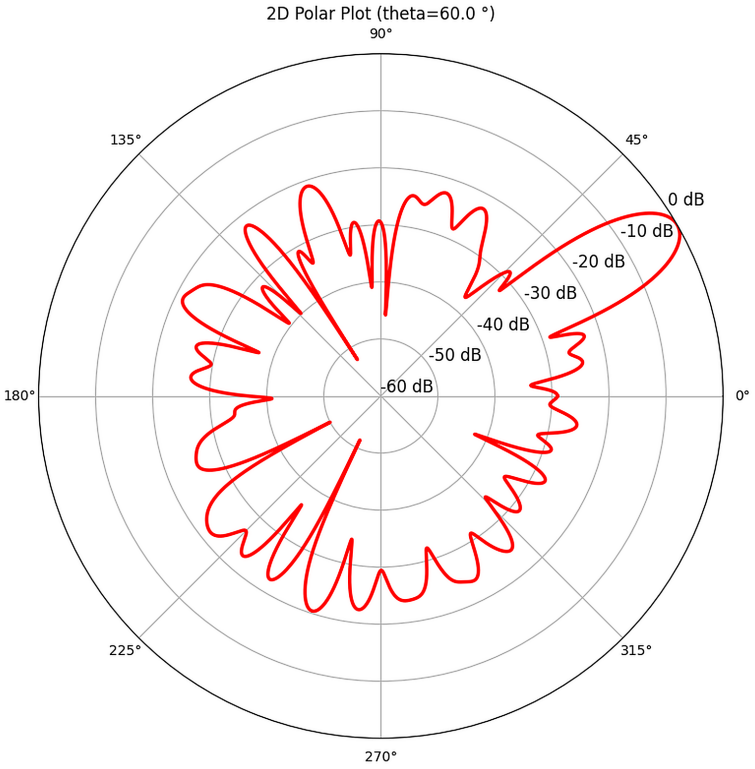


Figure 21. 2D radiation pattern of Design 2 with $\theta = 60^{\circ}$.

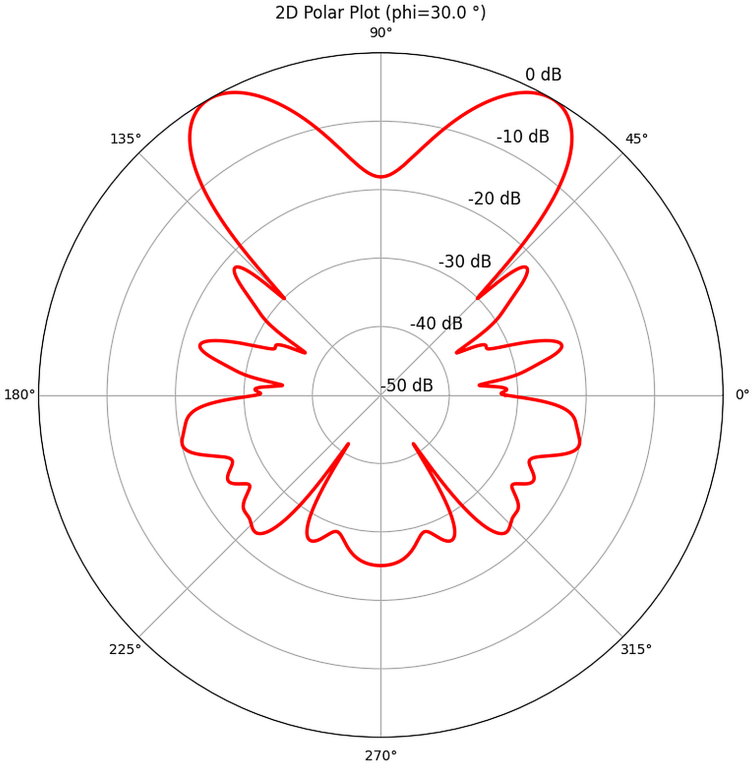


Figure 22. 2D radiation pattern of Design 2 with $\phi = 30^{\circ}$.

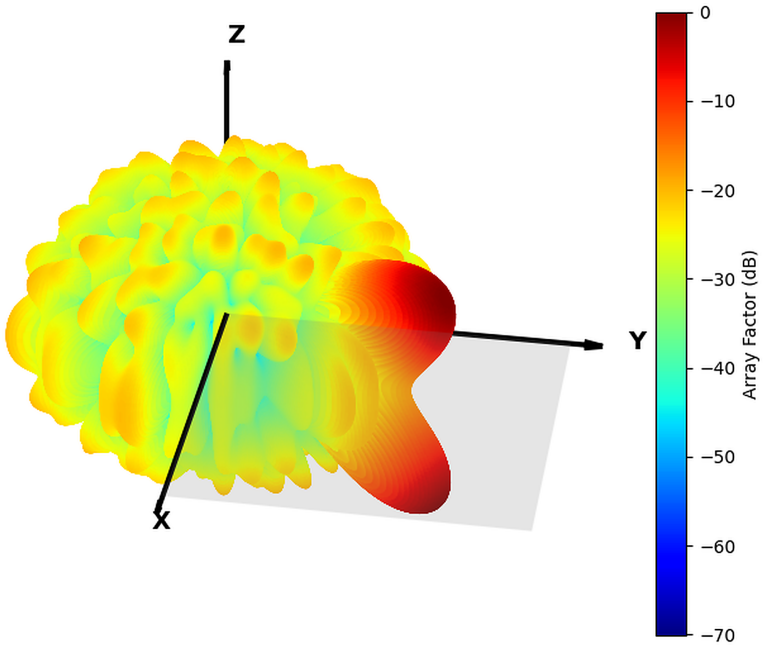


Figure 23. View of the 3D radiation pattern of Design 2.

5. Discussion

This study reexamines a challenge in adapting PSO from continuous to discrete search spaces. We demonstrate that the way randomness is incorporated into a Boolean velocity representation significantly influences exploration, stagnation, and the quality of the final solution. The suggested 0-mutation strategy that alters specific zeros to ones in the velocity bitstring modifies the practical interpretation of "limiting velocity" in Boolean PSO: rather than progressively inhibiting state transitions (as seen in $1 \rightarrow 0$ mutation), it maintains a regulated potential for bit flips in the XOR-based position update, thereby facilitating further exploration of the search space. This corresponds to the XOR truth table reasoning presented in the manuscript: an increased number of ones in the velocity bitstring immediately correlates with enhanced options for positional alterations, which is essential when the swarm faces the threat of premature convergence.

The findings from the benchmark suite support the primary assertion that mutation is not a trivial upgrade but an essential component for robust Boolean PSO performance. The no-mutation baseline (BPSO-CI) yields low success rates for numerous functions (Figure 4), indicating that particles become trapped in local optima when the swarm stabilizes in suboptimal areas. The introduction of mutation with a constant mutation factor (BPSO-CM-CI) results in a significant enhancement in accuracy across many functions (Figure 5). The manuscript's rationale for the $1 \rightarrow 0$ mutation is supported by the behavioral difference highlighted in the text: with the $1 \rightarrow 0$ mutation, optimal solutions are frequently attained prematurely (within approximately 100 iterations in the referenced example), indicating excessively rapid convergence and resulting stagnation. On the other hand, the $0 \rightarrow 1$ mutation facilitates sustained search activity throughout the duration of the run. In practical terms, 0-mutation seems to "re-initiate" search trajectories when the boolean velocity becomes excessively conservative, generating the necessary turbulence to avoid local optima.

A second consistent pattern is that time-varying parameters outperform stable values. A linearly changing mutation factor with constant inertia (BPSO-LM-CI) outperforms constant mutation/constant inertia (Figure 7 versus Figure 5), confirming the design intuition that heavy exploration is most valuable early, while reduced randomness later

promotes convergence. Changing inertia (BPSO-CM-LI) is beneficial (Figure 8), but not as much as changing mutation in this investigation. The combined schedule (BPSO-LM-LI) is the best-performing overall option, with both mutation and inertia varying linearly (Figure 9), leaving only a small number of functions outside the high-accuracy region. This shows that, in Boolean PSO, mutation scheduling is the primary driver of global search robustness, whereas inertia scheduling shapes the exploration-exploitation transition in a secondary but still beneficial way.

The statistical comparisons in various dimensionalities support this conclusion. BPSO-LM-LI has the best average Friedman ranking (Table 4) and scores first in 31 of 47 functions (Table 5). However, the Wilcoxon signed-rank test shows no statistically significant difference between BPSO-LM-LI and BPSO-LM-CI ($p = 0.906$), but BPSO-LM-LI outperforms the other comparable variations at the claimed significance level (Table 6). A practical view is that the mutation schedule captures the majority of the improvement, whereas inertia scheduling adds incremental advantages that may be problem-specific (or become more obvious under different dimensions or constraint types).

The same set of benchmark functions is used to conduct the statistical analysis comparing BPSO-LM-LI with four other population-based algorithms (BinPSO, BGWOA, BGA, and BDE). Once again, the proposed technique achieves the best average Friedman ranking (Table 7) and ranks first in 27 out of the 47 functions (Table 8). Moreover, the Wilcoxon signed-rank test demonstrates that BPSO-LM-LI outperforms three of the other algorithms at the established significance level (Table 9). The Binary Differential Evolution shows competitive performance, and the Wilcoxon analysis results in a p-value of 0.335, indicating little statistical difference between the algorithms. However, careful examination of Table 3 reveals that the two algorithms perform significantly differently in some functions, supporting the argument that the performance of the algorithms is problem-dependent. Ultimately, BPSO-LM-LI ranks first or second in 37 out of the 47 functions, providing a strong indication of the proposed approach's robustness.

Applying the BPSO-LM-LI variant to planar array thinning demonstrates that the approach is not only competitive on benchmarks but also capable of producing real-world engineering solutions under constraints. In a 16×16 array with $\lambda/2$ spacing, the method produced sparse excitations with significant element deactivation—164/256 (64% filling percentage)—of active elements in Design 1 and 160/256 (62.5% filling percentage) in Design 2. At the same time, the obtained results preserve good directivity (23.55 dBi and 21.19 dBi, respectively) and achieve *SLL* values very close to the -20 dB target (e.g., -19.79 dB and -19.69 dB). This supports the argument that the Boolean encoding is well-suited to thinning decisions (active/inactive) and that the proposed search dynamics can navigate a large combinatorial search space.

6. Conclusions

Based on the comprehensive evaluation across 47 benchmark functions and a demanding electromagnetics design problem, the following primary conclusions are drawn regarding the proposed 0-mutation BPSO approach:

- **Necessity of Mutation:** Incorporating mutation into the BPSO algorithm significantly increases the efficiency of the technique by providing sufficient perturbation to the particles.
- **Superiority of 0-Mutation:** The proposed mutation strategy, which involves converting zeros to ones ($0 \rightarrow 1$), is preferable to the reverse ($1 \rightarrow 0$), as it helps particles better escape local minima and avoid stagnation.
- **Impact of Parameter Scheduling:** Time-varying parameters (specifically the linear adaptation of mutation and inertia factors) demonstrate better performance than constant selection.

- **Competitive Robustness:** The optimally configured BPSO-LM-LI is shown to be statistically superior to the established population-based algorithms BinPSO, BGWOA, and BGA, while also achieving a slight overall advantage over BDE in a closely competitive comparison.
- **Real-World Applicability:** The 0-mutation BPSO has been successfully applied to a real-world high-dimensional planar antenna array thinning problem, achieving significant element reduction while maintaining directivity and side-lobe level constraints.

Author Contributions: Conceptualization, Z.D.Z.; methodology, E.G.N. and P.S.; software, E.G.N.; validation, Z.D.Z., P.I.L., and S.K.G.; formal analysis, Z.D.Z.; investigation, E.G.N. and Z.D.Z.; resources, P.S.; data curation, P.S.; writing—original draft preparation, E.G.N. and Z.D.Z.; writing—review and editing, Z.D.Z., P.I.L., and S.K.G.; visualization, P.S.; supervision, Z.D.Z. and S.K.G.; project administration, Z.D.Z.; funding acquisition, Z.D.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the European Union, through the Horizon Europe Marie Skłodowska-Curie Staff Exchanges Program "6G intelligent connectivity and interaction for users and infrastructures (6G-ICARUS)" under Grant 101131342.

Data Availability Statement: The code produced by this research can be found at https://github.com/manosnik/bpso_algorithm (accessed on 26 April 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kennedy, J.; Eberhart, R. Particle swarm optimization. In *Proceedings of the ICNN'95-International Conference on Neural Networks*; IEEE: New York, NY, USA, 1995; Volume 4, pp. 1942–1948.
2. Kennedy, J.; Eberhart, R. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*; IEEE: New York, NY, USA, 1997; Volume 5, pp. 4104–4108. [CrossRef]
3. Afshinmanesh, F.; Marandi, A.; Rahimi-Kian, A. A novel binary particle swarm optimization method using artificial immune system. In *Proceedings of the EUROCON 2005-The International Conference on "Computer as a Tool"*; IEEE: New York, NY, USA, 2005; Volume 1, pp. 217–220.
4. Shami, T.M.; El-Saleh, A.A.; Alswaitti, M.; Al-Tashi, Q.; Summakieh, M.A.; Mirjalili, S. Particle Swarm Optimization: A Comprehensive Survey. *IEEE Access* **2022**, *10*, 10031–10061. [CrossRef]
5. Zhang, Y.; Wang, S.; Ji, G. A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications. *Math. Probl. Eng.* **2015**, *2015*, 1–38. [CrossRef]
6. Wang, D.; Tan, D.; Liu, L. Particle swarm optimization algorithm: An overview. *Soft Comput.* **2018**, *22*, 387–408. [CrossRef]
7. Shu, X.; Liu, Y.; Liu, J.; Yang, M.; Zhang, Q. Multi-objective particle swarm optimization with dynamic population size. *J. Comput. Des. Eng.* **2023**, *10*, 446–467. [CrossRef]
8. Zhang, W.; Ma, D.; Wei, J.-J.; Liang, H.-F. A parameter selection strategy for particle swarm optimization based on particle positions. *Expert Syst. Appl.* **2014**, *41*, 3576–3584. [CrossRef]
9. Li, Z.; Ding, W.; Wang, X.; Li, W.; Seah, W.K. A Novel Exponential Dynamic Inertia Weight for Particle Swarm Optimization. In *Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN)*; IEEE: New York, NY, USA, 2024; pp. 1–8. [CrossRef]
10. Chen, Z.; Wang, Y.; Chan, T.H.T.; Li, X.; Zhao, S. A Particle Swarm Optimization Algorithm with Sigmoid Increasing Inertia Weight for Structural Damage Identification. *Appl. Sci.* **2022**, *12*, 3429. [CrossRef]
11. Arasomwan, M.A.; Adewumi, A.O. On the Performance of Linear Decreasing Inertia Weight Particle Swarm Optimization for Global Optimization. *Sci. World J.* **2013**, *2013*, 860289. [CrossRef]
12. Feng, Y.; Teng, G.F.; Wang, A.X.; Yao, Y.M. Chaotic Inertia Weight in Particle Swarm Optimization. In *Proceedings of the Second International Conference on Innovative Computing, Informatio and Control (ICICIC 2007)*; IEEE: New York, NY, USA, 2007; p. 475. [CrossRef]
13. Gao, Y.-L.; Duan, Y.-H. A New Particle Swarm Optimization Algorithm with Random Inertia Weight and Evolution Strategy. In *Proceedings of the 2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)*; IEEE: New York, NY, USA, 2007; pp. 199–203. [CrossRef]

14. Jian, W.; Xue, Y.-C.; Qian, J.-X. An improved particle swarm optimization algorithm with neighborhoods topologies. In *Proceedings of the 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*; IEEE: New York, NY, USA, 2004; Volume 4, pp. 2332–2337. [CrossRef]
15. Liu, Q.; Wei, W.; Yuan, H.; Zhan, Z.H.; Li, Y. Topology selection for particle swarm optimization. *Inf. Sci.* **2016**, *363*, 154–173. [CrossRef]
16. Liang, J.; Suganthan, P. Dynamic multi-swarm particle swarm optimizer. In *Proceedings of the 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005*; IEEE: New York, NY, USA, 2005; pp. 124–129. [CrossRef]
17. Han, W.; Li, H.; Gong, M.; Li, J.; Liu, Y.; Wang, Z. Multi-swarm particle swarm optimization based on CUDA for sparse reconstruction. *Swarm Evol. Comput.* **2022**, *75*, 101153. [CrossRef]
18. Yang, C.; Simon, D. A New Particle Swarm Optimization Technique. In *Proceedings of the 18th International Conference on Systems Engineering (ICSEng'05)*; IEEE: New York, NY, USA, 2005; pp. 164–169. [CrossRef]
19. Garg, H. A hybrid PSO-GA algorithm for constrained optimization problems. *Appl. Math. Comput.* **2016**, *274*, 292–305. [CrossRef]
20. Şenel, F.A.; Gökçe, F.; Yüksel, A.S.; Yiğit, T. A novel hybrid PSO–GWO algorithm for optimization problems. *Eng. Comput.* **2019**, *35*, 1359–1373. [CrossRef]
21. Chen, E.; Li, J.; Liu, X. In search of the essential binary discrete particle swarm. *Appl. Soft Comput.* **2011**, *11*, 3260–3269. [CrossRef]
22. Abbes, W.; Kechaou, Z.; Hussain, A.; Qahtani, A.M.; Almutiry, O.; Dhahri, H.; Alimi, A.M. An Enhanced Binary Particle Swarm Optimization (E-BPSO) algorithm for service placement in hybrid cloud platforms. *Neural Comput. Appl.* **2023**, *35*, 1343–1361. [CrossRef]
23. Al-kazemi, B.S.N.; Mohan, C.K. Multi-phase discrete particle swarm optimization. *Electr. Eng. Comput. Sci.* **2000**, *54*. Available online: https://surface.syr.edu/eecs/54/ (accessed on 13 March 2026).
24. Thaher, T.; Chantar, H.; Too, J.; Mafarja, M.; Turabieh, H.; Houssein, E.H. Boolean Particle Swarm Optimization with various Evolutionary Population Dynamics approaches for feature selection problems. *Expert Syst. Appl.* **2022**, *195*, 116550. [CrossRef]
25. Kumar, L.; Bharti, K.K. A novel hybrid BPSO–SCA approach for feature selection. *Nat. Comput.* **2021**, *20*, 39–61. [CrossRef]
26. Mirjalili, S.; Lewis, A. S-shaped versus V-shaped transfer functions for binary Particle Swarm Optimization. *Swarm Evol. Comput.* **2013**, *9*, 1–14. [CrossRef]
27. Guo, S.-S.; Wang, J.-S.; Guo, M.-W. Z-Shaped Transfer Functions for Binary Particle Swarm Optimization Algorithm. *Comput. Intell. Neurosci.* **2020**, *2020*, 1–21. [CrossRef]
28. Chen, Y.; Liu, J.; Zhu, J.; Wang, Z. An improved binary particle swarm optimization combing V-shaped and U-shaped transfer function. *Evol. Intell.* **2023**, *16*, 1653–1666. [CrossRef]
29. Islam, M.J.; Li, X.; Mei, Y. A time-varying transfer function for balancing the exploration and exploitation ability of a binary PSO. *Appl. Soft Comput.* **2017**, *59*, 182–196. [CrossRef]
30. Boeringer, D.; Werner, D. Particle Swarm Optimization Versus Genetic Algorithms for Phased Array Synthesis. *IEEE Trans. Antennas Propag.* **2004**, *52*, 771–779. [CrossRef]
31. Grimaccia, F.; Mussetta, M.; Niccolai, A.; Zich, R.E. Comparison of Binary Evolutionary Algorithms for Optimization of Thinned Array Antennas. In *Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC);* IEEE: New York, NY, USA, 2018; pp. 1–8. [CrossRef]
32. Jia, X.; Lu, G. A Hybrid Taguchi Binary Particle Swarm Optimization for Antenna Designs. *IEEE Antennas Wirel. Propag. Lett.* **2019**, *18*, 1581–1585. [CrossRef]
33. Marandi, A.; Afshinmanesh, F.; Shahabadi, M.; Bahrami, F. Boolean particle swarm optimization and its application to the design of a dual-band dual-polarized planar antenna. In *Proceedings of the 2006 IEEE International Conference on Evolutionary Computation*; IEEE: New York, NY, USA, 2006; pp. 3212–3218.
34. Robinson, J.; Rahmat-Samii, Y. Particle swarm optimization in electromagnetics. *IEEE Trans. Antennas Propag.* **2004**, *52*, 397–407. [CrossRef]
35. Gavana, A. Test Functions "Hardness", Global Optimization Benchmarks and AMPGO. 2013. Available online: http://infinity77.net/global_optimization/test_functions.html#test-functions-index (accessed on 20 December 2024).
36. Emary, E.; Zawbaa, H.M.; Hassanien, A.E. Binary grey wolf optimization approaches for feature selection. *Neurocomputing* **2016**, *172*, 371–381. [CrossRef]
37. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey Wolf Optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
38. Goudos, S. Antenna Design Using Binary Differential Evolution: Application to discrete-valued design problems. *IEEE Antennas Propag. Mag.* **2017**, *59*, 74–93. [CrossRef]
39. Computational Optimization and Innovation (COIN) Laboratory, Michigan State University. Codes. Available online: http://www.egr.msu.edu/~kdeb/codes.shtml (accessed on 13 March 2026).
40. Wang, L.; Fu, X.; Mao, Y.; Ilyas Menhas, M.; Fei, M. A novel modified binary differential evolution algorithm and its applications. *Neurocomputing* **2012**, *98*, 55–75. [CrossRef]

41. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]
42. Balanis, C.A. *Antenna Theory: Analysis and Design*; John Wiley & Sons: Hoboken, NJ, USA, 2016.
43. Yager, R.R. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybern.* **1988**, *18*, 183–190. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.