Article

An Enhanced Particle Swarm Optimization (PSO) Algorithm Employing Quasi-Random Numbers

Shiva Kumar Kannan [1] and Urmila Diwekar [2,*]

[1] Ridge High School, Basking Ridge, Bernards, NJ 07920, USA; shivakannankumar@gmail.com
[2] Vishwamitra Research Institute, Crystal Lake, IL 60012, USA
* Correspondence: urmila@vri-custom.org; Tel.: +1-(630)-886-3047

Abstract: This paper introduces an innovative Particle Swarm Optimization (PSO) Algorithm incorporating Sobol and Halton random number samplings. It evaluates the enhanced PSO's performance against conventional PSO employing Monte Carlo random number samplings. The comparison involves assessing the algorithms across nine benchmark problems and the renowned Travelling Salesman Problem (TSP). The results reveal consistent enhancements achieved by the enhanced PSO utilizing Sobol/Halton samplings across the benchmark problems. Particularly noteworthy are the Sobol-based PSO improvements in iterations and the computational times for the benchmark problems. These findings underscore the efficacy of employing Sobol and Halton random number generation methods to enhance algorithm efficiency.

Keywords: enhanced PSO; SOBOL; Halton; quasi-random numbers

check for updates

Citation: Kannan, S.K.; Diwekar, U. An Enhanced Particle Swarm Optimization (PSO) Algorithm Employing Quasi-Random Numbers. *Algorithms* **2024**, *17*, 195. https://doi.org/10.3390/a17050195

Academic Editors: Łukasz Knypiński and Roberto Montemanni

Received: 6 March 2024
Revised: 26 April 2024
Accepted: 30 April 2024
Published: 6 May 2024

1. Introduction

Particle Swarm Optimization (PSO) is a metaheuristic algorithm for optimization problems. PSO was first introduced in 1995 by James Kennedy and Russell Eberhart [1]. The algorithm is based on the concept of social behavior, where particles (potential solutions) move towards the optimal solution through interactions with other particles in the search space. PSO has been widely used in various fields, including engineering, science, and finance, due to its simplicity, robustness, and efficiency. Despite its success, PSO suffers from several limitations. One of the main limitations is its slow convergence rate, which can be attributed to the premature convergence [2] of the particles towards local optima. This issue can be addressed by introducing efficient improvement techniques in PSO. Several enhancement ideas have been proposed in the past to improve the convergence rate of the PSO algorithm, and they are listed below. Firstly, the inertia weight technique was suggested by Russell Eberhart and Ying Shi [3]. The inertia weight technique is a well-known approach for enhancing the convergence speed of PSO. The inertia weight is used to control the movement of particles in the search space. The idea is to maintain a balance between exploration and exploitation of the search space. The inertia weight is updated at each iteration based on a predefined formula, which controls the speed and direction of particle movement. Various formulas have been proposed for updating the inertia weight, such as linear, nonlinear, and adaptive. The choice of the inertia weight formula depends on the optimization problem and the PSO parameters. Second, the concept of a mutation operator was proposed [4]. A mutation operator is a powerful tool for enhancing the diversity of the PSO population. The mutation operator randomly modifies the position of a particle to generate a new solution in the search space. This operation can prevent premature convergence by introducing new solutions that may lead to better solutions. The mutation operator can be applied at different stages of the PSO algorithm, such as before or after the velocity update. Similar to the mutation operator, another operator known as the crossover operator has also been applied to the PSO algorithm [5–7]. This concept involves the mixture of the attributes of different solutions to gain exploration and to achieve high diversity in potential results to avoid premature convergence. Third, the opposition-based learning technique was suggested [8]. Opposition-based learning (OBL) is a technique that uses the opposite of the current best solution to generate new solutions. The idea behind OBL is that the opposite of the best solution may represent a good direction for exploration in the search space. OBL can improve the diversity and convergence speed of PSO by generating new solutions that are different from those of the current population. Fourth, hybridization with other metaheuristics has been proposed [9]. Hybridization with other metaheuristics is a common approach for improving the efficiency of PSO. The idea is to combine the strengths of different metaheuristics to overcome their weaknesses. For example, PSO can be combined with genetic algorithms (GA), simulated annealing (SA), or ant colony optimization (ACO). The hybridization approach can enhance the exploration and exploitation capabilities of PSO, leading to better solutions in less time. Fifth, dynamic parameter tuning was presented [9]. The PSO parameters, such as the swarm size, maximum velocity, and acceleration coefficients, significantly impact the algorithm's performance. Dynamic parameter tuning is a technique that adjusts the PSO parameters based on the search history during the optimization process. The idea is to adapt the PSO parameters to the problem characteristics and the search progress to improve the convergence speed and solution quality. In conclusion, efficient improvement techniques in PSO can enhance the algorithm's convergence speed and solution quality. The approaches discussed in this paper [9], including the inertia weight technique, mutation operator, opposition-based learning, hybridization with other metaheuristics, and dynamic parameter tuning, can be used individually or in combination to address the limitations of PSO. Tareq M. Shami and a team [10] of researchers conducted a comprehensive survey on PSO. The survey discusses techniques such as varying the inertia weight and hybridizations, which are discussed above. The survey also states that the ability of PSO to be hybridized with other optimization algorithms has contributed to its popularity. Another technique described in the survey is velocity clamping [10], a technique introduced by Eberhart and Kennedy. Velocity clamping involves setting bounds for the values of the velocities of the particles in all the dimensions. Another approach to improving the efficiency of the PSO algorithm discussed in this paper [10] is varying the controlling parameters, such as using the varying inertia weight technique in which inertia weight changes throughout the optimization process, or acceleration coefficient techniques in which the two constant controlling parameters for PSO other than the inertia, are chosen in different ways to yield optimal solutions while evading premature convergence. Many other approaches have been discussed both in the survey and elsewhere. The choice of approach depends on the problem characteristics and the available computational resources. However, most of these approaches can provide problem-dependent solution methods. In this paper, we proposed a new approach to replace the random numbers used for this method with quasi-random numbers [11–13], like Halton and Sobol, by maintaining the k-dimensional uniformity of these quasi-random numbers. This not only provides a generalized approach to any optimization problem, but this method can be used in conjunction with the earlier enhancement techniques like the inertia weight technique, mutation operator, opposition-based learning, hybridization with other metaheuristics, and dynamic parameter tuning. In this research, two enhanced versions of PSO (one using Sobol random numbers and the other using Halton random numbers) were proposed with the intention of speeding up the convergence of the standard PSO algorithm. To test the efficiency improvement of the two proposed enhancements of the standard PSO algorithm, the number of iterations taken to achieve the optimum of the well-known cigar, ellipsoid, and paraboloid functions [14], along with the number of iterations taken to obtain an optimal path for the famous Travelling Salesman Problem (TSP) [15–18], were noted. Following this, improvement in terms of the optimum of the objective function and the number of iterations needed to reach the global optimum were calculated for both the PSO enhanced with Sobol random number samplings and the PSO enhanced with Halton random number samplings, with respect to the standard PSO, which uses Monte Carlo random number samplings. All the results for each benchmark function and TSP unanimously show efficiency improvement due to the use of the Sobol and Halton sequences. Additionally, we noted that the more decision variables an optimization problem has, the improvement due to Sobol and Halton sequences increases. In conclusion, both the enhancements of the standard PSO presented in this research, one utilizing Sobol random numbers and the other utilizing Halton random numbers, consistently show efficiency improvement and a better optimum, meaning that they successfully have increased the speed of convergence of the standard PSO algorithm.

In addition, we have also compared our enhanced PSO with the SALP meta-heuristic variant [19]. This is a recent algorithmic approach developed to improve the convergence rate. Our enhancement is compared with the SALP to see whether the approach of avoiding clusters in random number generation can make the enhanced PSO algorithm perform better than the SALP algorithm. The algorithms are compared for the four benchmark problems.

The time-varying inertia factor was introduced to improve the converge performance of the PSO [20,21]. The authors in [22] introduced a time-varying acceleration coefficient in addition to the time-varying inertia factor. They introduced a PSO concept called a self-organizing hierarchical particle swarm optimizer with a time-varying acceleration coefficient. For the velocity update in PSO, only the social part and cognitive part were considered, and to avoid stagnation in the search space, a time-varying mutation step size was used as well.

An adaptive particle swarm optimization (APSO) that features better search efficiency than the classical particle swarm optimization (PSO) is presented in [23]. It was engineered to perform a global search over the entire search space with faster convergence speed. The APSO algorithm was carried out in two main steps. In the first step, the algorithm evaluated the population distribution and particle fitness based on which a real-time evolutionary state was estimated. This enabled the automatic control of inertia weight, acceleration coefficients, and other algorithmic parameters in real-time to improve the search efficiency and convergence speed in the subsequent step.

The multimodal PSO approach proposed in [24] addressed the issues associated with poor local search ability and the requirement of prior knowledge to initialize the PSO algorithm parameters. The authors in [24] proposed an optimizer based on a distance-based locally informed PSO variant. The algorithm eliminated the need to specify the *niche* parameters in the PSO and enhanced its fine-searching capabilities. To guide the search direction of the particles, each particle used local best information instead of the global best as in the conventional PSO, and the neighborhood of the particle was measured using the Euclidean distance to perform the local best search.

In [25], a hybrid algorithm that combined particle swarm optimization with simulated annealing behavior (SA-PSO) was proposed. The SA-PSO algorithm takes advantage of the good solution quality provided by the simulated annealing and fast searching ability inherent to the particle swarm optimization. It was concluded that the hybrid algorithm could have higher efficiency, better quality and faster convergence speed than conventional PSO variants.

An economic environmental hydrothermal scheduling problem classified as a multi-objective nonlinear constrained optimization was solved using PSO [26]. The algorithm adopted an elite archive set to conserve Pareto optimal solutions and provide multiple evolutionary directions for individuals, while neighborhood searching and chaotic mutation strategies enhance the search capability and diversity of the population. The PSO algorithm also incorporated a constraint handling scheme designed to adjust the constraint violation of hydro and thermal plants.

To avoid parameter selection and overcome the premature convergence problem in the PSO optimization, an adaptive fuzzy particle swarm optimization based on a fuzzy inference system, which incorporated a variable neighborhood search strategy and hybrid evolution, was proposed in [27] and applied to the parameter estimation of nonlinear Hydro Turbine Regulation Systems. The results concluded that the new algorithm's parameter error and the objective function were significantly smaller than the other algorithms. The estimated model could accurately reflect the dynamic characteristics of the real system, proving that the fuzzy PSO variant was an effective and efficient parameter estimation method.

Support vector machine (SVM)-based classifiers need accurate parameter estimation for high-accuracy classification, and in [28], an improved variant of the PSO is used to estimate the SVM parameters. Specifically, dynamic adjustment of inertia is proposed to optimize the parameters of the SVM. The computation of the inertia weight in the PSO proposed in [28] involves the nonlinear reduction in the inertia weight as the number of iterations increases. In particular, the introduction of random function inertia weight in the PSO avoids falling into the local extremum and, at the same time, increases diversity and the global searching ability during the optimization process.

In addition to the above-mentioned PSO algorithm comparisons, we compare the iteration efficiency of the enhanced PSO approach employing the Sobol sequence using the random inertia PSO variant for the benchmark problems.

2. Materials and Methods

Particle Swarm Optimization

There are countless examples of swarms in the real world, ranging from flocking birds to hunting wolves. When searching for nourishment, the individuals of these swarms, called particles to understand PSO, begin random exploration and then start gravitating towards the findings of other swarm individuals. While following signs of nourishment and searching randomly in space, these particles move towards their objective through knowledge of their discoveries and discoveries of the swarm. Similarly, in the PSO algorithm, proposed decision variable sets gravitate towards the optimal findings of each other, themselves, and the whole swarm together to achieve the globally optimal set of decision variables, yielding the optimal function value. The sets of decision variables are called position vectors. They are updated by vectors called velocity vectors (based on the physics principle that change in position is proportional to the velocity), by randomness, and by attraction towards their personal best sets of decision variables and the unanimous global best set of decision variables found by the whole swarm. The global optimum of the function is achieved through gradual movement towards optimal findings of the swarm. The following are the steps of the PSO algorithm.

- Initialize Parameters:
    - Define the population size (number of particles), Np; Define the number of decision variables (dimension), D;
    - Define the maximum number of iterations, T;
    - Define the inertia weight, ω;
    - Define acceleration constants: cognitive and social, c_1, c_2;
    - Initialize the position and velocity of each particle randomly within the search space.

The initially proposed set of solutions is grouped together to form the population matrix, which is written as follows:

$$\begin{bmatrix} x_0^{00} & \cdots & x_0^{0(D-1)} \\ \vdots & \ddots & \vdots \\ x_0^{(Np-1)0} & \cdots & x_0^{(Np-1)(D-1)} \end{bmatrix}$$

Each row in this matrix represents a potential decision variable vector intended to optimize the objective function. Each element in a row is the position with respect to a particular dimension of the particle that corresponds with that row. The subscript for each element is 0, as these values are the initial values in the matrix (0th iteration).

The velocity matrix can be represented as follows:

$$\begin{bmatrix} v_0^{00} & \cdots & v_0^{0(D-1)} \\ \vdots & \ddots & \vdots \\ v_0^{(Np-1)0} & \cdots & v_0^{(Np-1)(D-1)} \end{bmatrix}$$

This represents the velocity of each particle in all the dimensions. The velocity matrix changes the position matrix, and that is inspired by the physics concept that displacement is proportional to velocity. The superscript j is the index of the dimension.

- Set the best-known position for each particle n, P_n^{ij} to its initial position.

$$P^j = x^{0j}$$

- Evaluate Fitness:

  Evaluate the fitness (objective function value) for each particle based on its current position.
- Update the personal best position for each particle G_n^j if its current fitness is better than its previous best fitness.
- Update Global Best:
  - Determine the particle with the best fitness among all particles in the swarm, P_n^*.
  - Update the global best position with the particle's position with the best fitness, G_n^*.
  - Update Velocities and Positions:

  For each particle, update its velocity and position based on the following formulae.

  The velocity of each particle at iteration n is updated according to the equation given below.

$$v_{n+1}^{ij} = \omega \times v_n^{ij} + c_1 \times r_1 \times \left(P_n^j - X_n^{ij}\right) + c_2 \times r_2 \times \left({G_n}^j - X_n^{ij}\right) \quad (1)$$

where r_1 and r_2 are random numbers. So, we have two random numbers per variable.

The corresponding position is updated according to the following equation:

$$x_{n+1}{}^{ij} = x_n{}^{ij} + v_{n+1}^{ij} \quad (2)$$

- Check Stopping Criteria:
  - If the maximum number of iterations is reached or a satisfactory solution is found, stop the algorithm.
  - Otherwise, go back to step 2 and repeat the process.
- Output:

  Return the global best position as the solution to the optimization problem.

  Quasi-random sequence enhancements

In this work, we hypothesize that the success of PSO depends on the choice of appropriate random samples. At each iteration of PSO, two random numbers are generated for each decision variable, as shown in Equation (1). These random numbers are computer-generated random numbers and are called pseudorandom numbers or Monte Carlo random numbers. A sequence of random numbers must have two essential properties: uniformity, i.e., they are equally probable everywhere, and independence, i.e., the current value of a random variable has no relation with the previous values. Figure 1 shows the two random variables generated using a computer (Monte Carlo or pseudorandom numbers) with samples equal to 100. As seen in the figure, for uniformity, the points should be equally distributed; that is not the case here. We need more samples to cover the points equally in that 2-dimensional space. This means more iterations of the algorithm. To circumvent this problem and to increase the efficiency of PSO, we are presenting a construct based on quasi-random numbers. Some well-known quasi-random sequences are Halton, Hammersley, Sobol, Faure, Korobov, and Neiderreiter [11–13]. The choice of an appropriate quasi-Monte Carlo sequence is a function of discrepancy. Discrepancy is a quantitative measure of the deviation of the sequence from the uniform distribution. Therefore, it is desirable to choose a low discrepancy sequence. The Halton, SOBOL, and Hammersley are some examples of low-discrepancy sequences. Here, we are working with the two sequences, Halton and SOBOL, as described below.

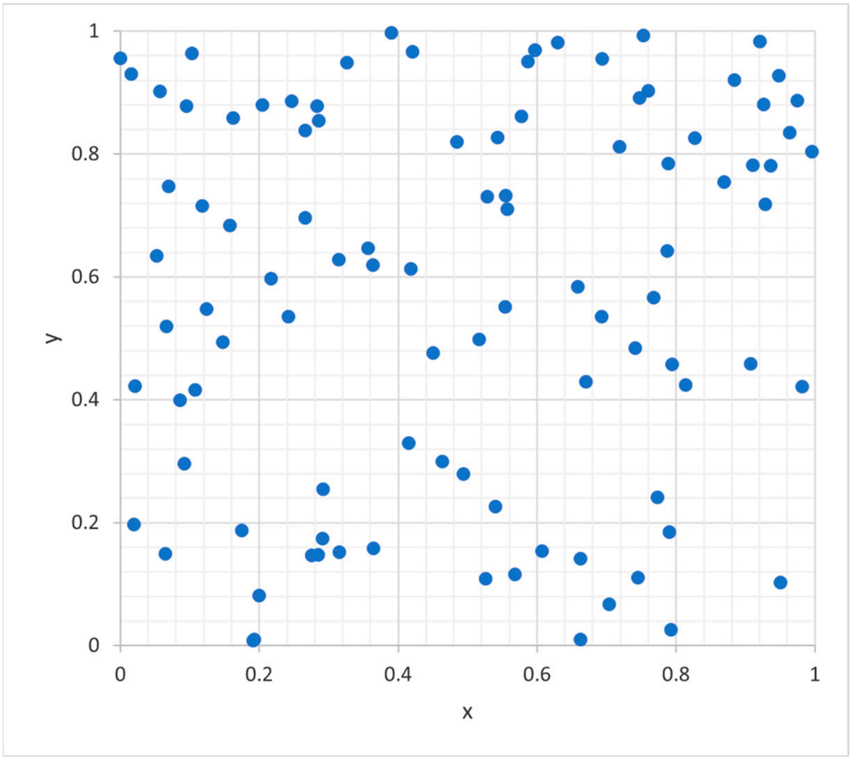

Figure 1. Two-dimensional pseudorandom numbers (100 points).

Halton Sequence Points:

The design of Halton points is given below. Any integer n can be written in radix-R notation (R is an integer) as follows:

$$n \equiv n_m n_{m-1} \ldots n_2 n_1 n_0 \tag{3}$$

$$n = n_0 + n_1 R + n_2 R^2 + \ldots + n_m R^m \tag{4}$$

where $m = [\log_R n] = [\ln n / \ln R]$ (the square brackets denote the integral part). A unique fraction between 0 and 1 called the inverse radix number can be constructed by reversing the order of the digits of n around the decimal point as follows:

$$\varphi_R(n) = n_0 n_1 n_2 \ldots n_m = n_0 R^{-1} + n_1 R^{-2} + \ldots + n_m R^{-m-1} \tag{5}$$

The Halton points on a k-dimensional cube are given by the following sequence:

$$\vec{z_k}(n) = \left(\varphi_{R_1}(n), \varphi_{R_2}(n), \ldots, \varphi_{R_{k-1}}(n)\right),\ n = 1, 2, \ldots, N+1 \tag{6}$$

where $R_1,\ R_2, \ldots R_{k-1}$ are the first $k-1$ prime numbers. The Halton points are $\vec{x_k}(n) = 1 - \vec{z_k}(n)$.

Figure 2 shows 2-dimensional Halton points (100 samples), which shows better uniformity than Figure 1.

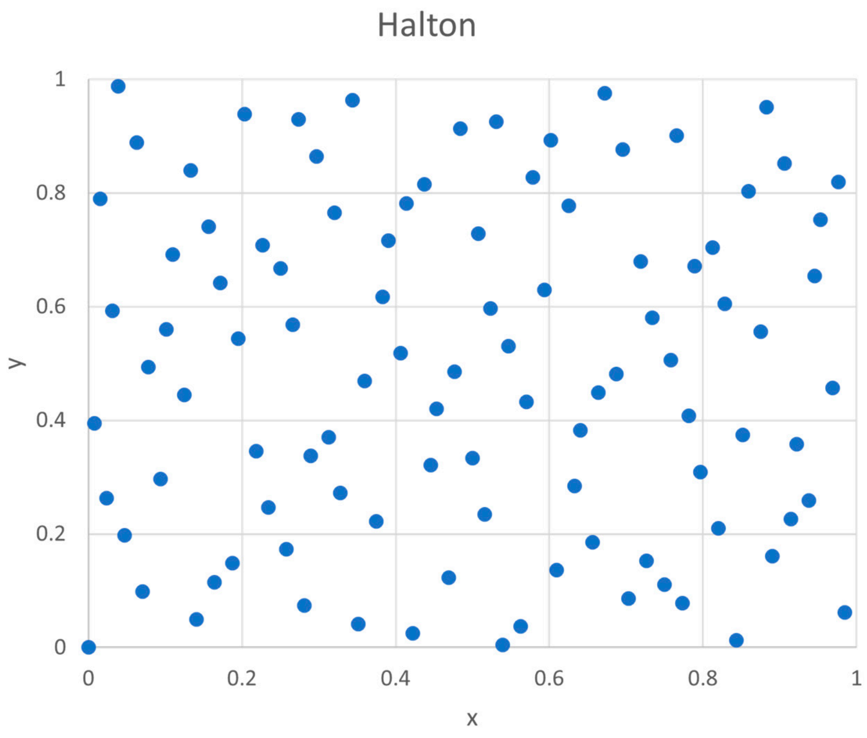

Figure 2. Two-dimensional Halton Points (100).

SOBOL Sequence Points:

Like many other quasi-random sequences, the SOBOL sequence starts from the Van der Corput sequence in base 2. To generate the Vander Corput sequence, consider the k-th point in the Sobol sequence; this integer k can be written as a linear combination of a nonnegative power of base 2 as follows.

$$k = a_0(k) + 2a_1(k) + 2^2 a_2(k) + \ldots + 2^r a_r(k) \tag{7}$$

where r is a large number.

Then, the k^{th} element in the Sobol sequence is given by the following equation:

$$x^k = 1/2y_1(k) + 1/2^2 y_2(k) + \ldots + 1/2^r y_r(k) \tag{8}$$

where the coefficients $y_i(k)$ can be obtained using the following expression:

$$\begin{Bmatrix} y_1(k) \\ y_2(k) \\ \ldots \\ y_r(k) \end{Bmatrix} = \mathrm{V} \begin{Bmatrix} a_0(k) \\ a_1(k) \\ \ldots \\ a_{r-1}(k) \end{Bmatrix} \mathrm{mod2} \tag{9}$$

where V is the generation matrix whose elements are 0 or 1. V is an identity matrix for the Vander Corput sequence.

The operation in Equation (9) can be represented as follows:

$$a_0(k)V_1 \oplus a_1(k)\boldsymbol{V}_2 \oplus a_2(k)v_3 \ldots a_{r-1}(k)V_r$$

where V_i is an element of V and $\oplus$***d***enotes binary addition.

The calculation of generation matrix V involves primitive polynomial A, primitive polynomial of degree d and all the coefficients A1 to Ad−1, which are either 1 or 0 and are given below.

$$P = X^d + A_1 X^{d-1} + \ldots + A_{d-1} X + 1 \tag{10}$$

Direction vectors M_i are generated by the recursive equation given below for i > d, and the initial direction vectors, i.e., for i < d, are generated by selecting an odd integer between 0 and 2^d.

$$M_i = 2^1 A_1 M_{i-1} \oplus 2^2 A_2 M_{i-2} \oplus \ldots\ldots\ldots \oplus 2^{d-1} A_{d-1} M_{i-d+1} \oplus 2^d M_{i-d} \oplus M_{i-d}. \tag{11}$$

Then, the generation matrix elements can be generated as shown below.

$$V_i^j = \frac{M_i}{2^i} \tag{12}$$

Thus, SOBOL sequence can be generated by generating the V matrix and the Van der Corput sequence in base 2. Figure 3 shows that SOBOL points show better uniformity than pseudo-random numbers from the computer (Figure 1).

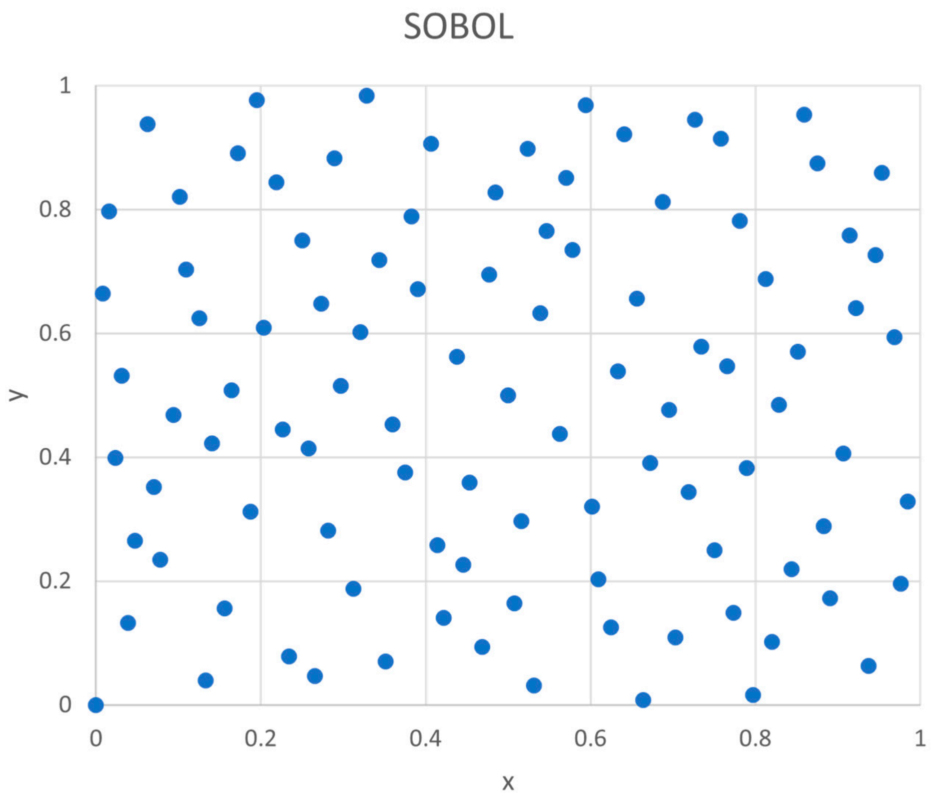

Figure 3. Two-dimensional SOBOL points (100).

We show where the random numbers are used in PSO with traditional PSO in Figure 4 and enhanced PSO with Halton or SOBOL in Figure 5. However, to maintain k-dimensional uniformity, the Halton and SOBOL points cannot be generated one at a time; they must be generated together with the whole sample (for all iterations).

It should be noted that all other efficiency enhancements proposed in the literature can be directly applicable to our new algorithms as we are only changing the random numbers.

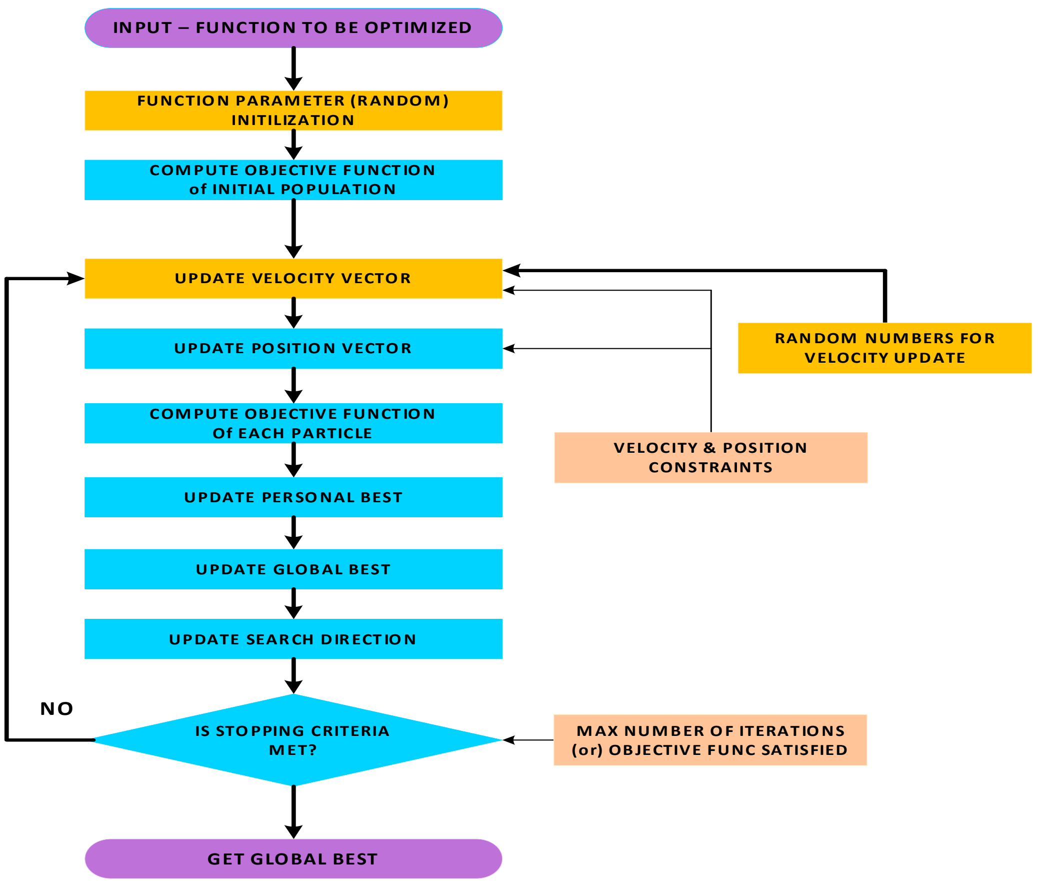
INPUT – FUNCTION TO BE OPTIMIZED
FUNCTION PARAMETER (RANDOM) INITILIZATION
COMPUTE OBJECTIVE FUNCTION of INITIAL POPULATION
UPDATE VELOCITY VECTOR
RANDOM NUMBERS FOR VELOCITY UPDATE
UPDATE POSITION VECTOR
COMPUTE OBJECTIVE FUNCTION Of EACH PARTICLE
VELOCITY & POSITION CONSTRAINTS
UPDATE PERSONAL BEST
UPDATE GLOBAL BEST
UPDATE SEARCH DIRECTION
NO
IS STOPPING CRITERIA MET?
MAX NUMBER OF ITERATIONS (or) OBJECTIVE FUNC SATISFIED
GET GLOBAL BEST

Figure 4. Traditional PSO flowchart.

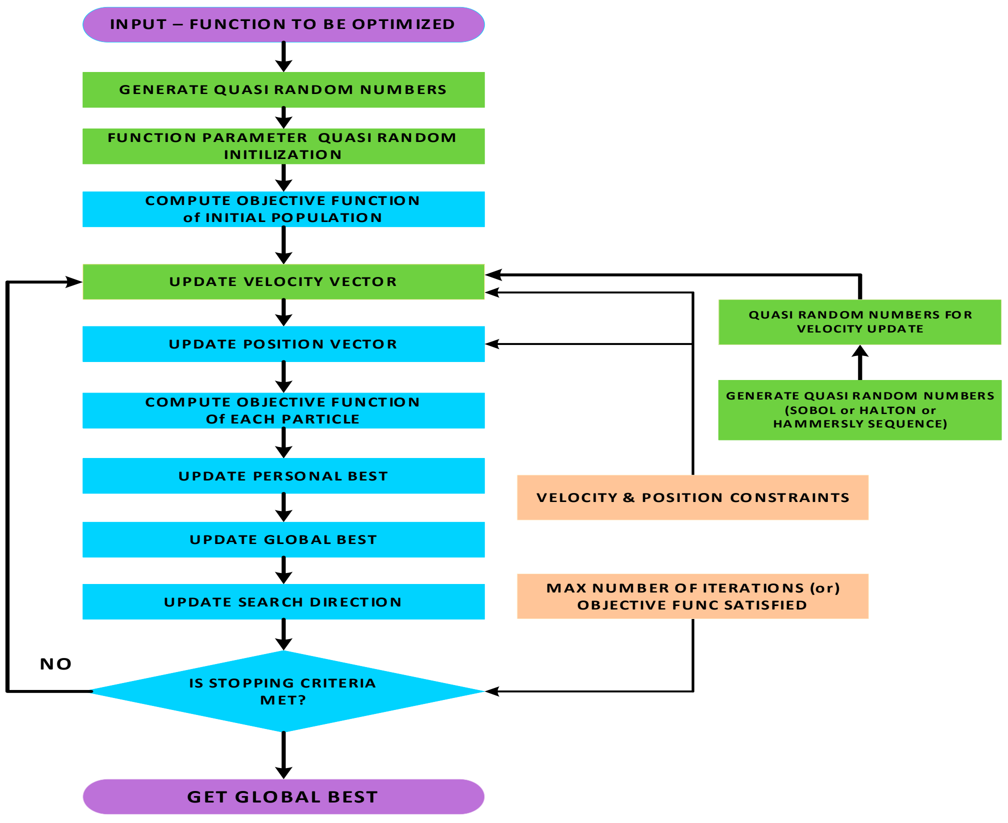
INPUT – FUNCTION TO BE OPTIMIZED
GENERATE QUASI RANDOM NUMBERS
FUNCTION PARAMETER QUASI RANDOM INITILIZATION
COMPUTE OBJECTIVE FUNCTION of INITIAL POPULATION
UPDATE VELOCITY VECTOR
QUASI RANDOM NUMBERS FOR VELOCITY UPDATE
UPDATE POSITION VECTOR
GENERATE QUASI RANDOM NUMBERS (SOBOL or HALTON or HAMMERSLY SEQUENCE)
COMPUTE OBJECTIVE FUNCTION Of EACH PARTICLE
UPDATE PERSONAL BEST
VELOCITY & POSITION CONSTRAINTS
UPDATE GLOBAL BEST
UPDATE SEARCH DIRECTION
MAX NUMBER OF ITERATIONS (or) OBJECTIVE FUNC SATISFIED
NO
IS STOPPING CRITERIA MET?
GET GLOBAL BEST

Figure 5. Enhanced PSO flowchart.

3. Results

Test Cases

To perform simulation and to estimate the standard deviations and the computation times, all the following algorithms were on a Windows 10 laptop equipped with an i7 core processor and with inbuilt 16 GB RAM. The simulations were run for 50 iterations and averaged to estimate the elapsed time. The computation elapsed times were estimated using the times associated with the start time and end time of the algorithm execution. These elapsed times were averaged across multiple runs and were used to measure the average elapsed time. The mean values for the number of iterations were estimated and rounded to provide the number of iterations in the following tables. All PSO variants implemented the mutation scheme that used uniformly generated random numbers.

To test the efficiency improvements of the enhanced PSO suggested in this paper, both mixed and continuous versions of three well-known benchmark functions, namely the Cigar, Ellipsoid, and Paraboloid functions, along with the famous Travelling Salesman Problem, were taken as test cases (Table 1). Three algorithms consisting of one PSO code with no enhancement (using Monte Carlo random number samplings), a second PSO code that uses Sobol random number samplings, and a third PSO code that uses Halton random numbers each ran for 5, 10, 15, and 20 decision variables to optimize both the mixed and continuous versions of the three benchmark functions and to solve the TSP problem. The number of iterations taken to achieve an optimum is recorded for all three algorithms to reach the global optimum.

Table 1. Test cases for algorithm testing [11].

	Function	Formula	Range
Continuous Optimization Problems			
1	Cigar	$f(x) = {x_1}^2 + 10^4 \sum_{i=2}^{NC} {x_i}^2$	$[-3,\ 3]^{NC}$
2	Parabolic	$f(x) = \sum_{i-1}^{NC} {x_i}^2$	$[-3,\ 3]^{NC}$
3	Ellipsoid	$f(x) = \sum_{i-1}^{NC} 5^{\frac{i-1}{n-1}} {x_i}^2$	$[-3,\ 3]^{NC}$
Mixed-Variable Combinatorial Optimization Problems			
4	Cigar	$f(x,\ y) = {x_i}^2 + 10^4 \sum_{i=1}^{ND} {x_i}^2 + {y_1}^2 + 10^4 \sum_{i=2}^{ND} {y_i}^2$	$[-3,\ 3]^{NM}$
5	Parabolic	$f(x,\ y) = \sum_{i=1}^{NC} {x_i}^2 + \sum_{i=1}^{ND} {y_i}^2$	$[-3,\ 3]^{NM}$
6	Ellipsoid	$f(x,\ y) = \sum_{i=1}^{NC} 5^{\frac{i-1}{n-1}} {x_i}^2 + \sum_{i=1}^{ND} 5^{\frac{i-1}{n-1}} {y_i}^2$	$[-3,\ 3]^{NM}$

Traveling Salesman Problem Optimization Procedure

The Traveling Salesman Problem is a discrete combinatorial optimization problem [12–15]. The locations of many cities are given, and an optimal order in which the cities are traversed is calculated. A position vector is the suggested order of cities. The velocity vector is a sequence of two-element tuples in which each tuple consists of two indices of elements to be swapped to make the path more optimal. For example, if there are five cities labeled with indices {1, 2, 3, 4, 5}, the population matrix could consist of different suggested orders in which they are to be traversed, such as {2, 4, 3, 5, 1} and {5, 4, 3, 1, 2}. The velocity vector for the first suggested order of cities could be {(1, 2), (3, 2), (4, 5)}, while for the second, it could be {(5, 1)}. In this case, the first element would be swapped with the second, the third with the second after that, and the fifth with the fourth after that, in the first suggested sequence of cities. For the second, the fifth and first elements are to be swapped.

ω, α, and β are pre-determined constants used in this algorithm. A random number is generated for each swap in the previous velocity vector. For each random number that is less than ω, the corresponding swap is included in the new velocity vector. Similarly, this process is carried out for the second term and the third term with α and Beta instead of ω. These three random numbers are generated together to maintain the k-dimensional uniformity of the quasi-random number sequence when Halton or SOBOL is used. Through the usage of this swapping method, the optimal order in which the cities must be visited is attained.

For the i^{th} particle at the iteration index n, to update the velocity, the following equation can be used.

$$th:$$

$$v_{n+1}^{i} = \{\omega * v_n^i\} \bigoplus \{\alpha * (P_n^i - X_n^i)\} \bigoplus \left\{\beta * \left(G_n^j - X_n^i\right)\right\} \tag{13}$$

$Hence,\ v_{n+1}^{i}\ is\ a\ set\ of\ swaps.$

Here, the $\oplus$ is the merging operator, which merges sequences of swaps into a new swap sequence.

$$\alpha < 1, \beta < 1\ and\ \omega < 1 \tag{14}$$

For the i^{th} particle at the iteration index n, we can apply the new updated velocity v_{n+1}^{i} to the current position X_n^i, (which is a set of node sequences or a node list in TSP) and obtain the updated position X_{n+1}^{i}.

$$X_{n+1}^{i} = X_n^i \xleftarrow{Apply\ Swaps} v_{n+1}^{i} \tag{15}$$

For $\omega * v_n$, if v_n is a vector of L elements to begin with, then L random numbers are generated and for each random number that is less than ω, the corresponding swap in v_n is used. $\left(P_n^i - X_n^i\right)$ is a swap sequence to move from X_n^i to P_n^i. For example, if P_n^i is {3, 4, 5, 2, 1} and $X_n^i\ is\ \{5,3,4,1,2\}$, the set of swaps needed to move fromX_n^i to P_n^i is {(1, 3), (3, 2), (4, 5)} (assuming that indices start from 1).

$\alpha * \left(P_n^i - X_n^i\right)$ is a set of swaps that are selected from the swap sequence vector $\left(P_n^i - X_n^i\right)$ of length N based on the N random numbers generated that are less than α.

During initialization, for each particle i, X_0^i is set to a random selection of cities whose IDs are the city node index. If there are N cities to be visited, then X_0^i is a vector of length N.

The following are tables comparing the results produced by Monte Carlo random number sampling with Sobol random number sampling and Halton random number sampling.

Sobol vs. Monte Carlo Random Numbers

We compare the performance of using SOBOL versus conventional random number approaches based on PSO for the three functions (continuous and mixed variable form) in Tables 2–7. In addition, we also compare the SALP algorithm for the continuous ellipse and paraboloid and mixed variable ellipse and paraboloid function optimizations.

Table 2. Comparison of performances of both Monte Carlo and Sobol random number samplings in PSO for the continuous variable Cigar function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	85	107	1.9	5.1	0.31	0.37	21%
10	167	193	2.4	6.3	0.46	0.68	14%
15	183	312	5.2	30.5	0.69	1.1	42%
20	231	447	26.5	30.1	0.87	1.6	49%

Table 3. Comparison of performances of both Monte Carlo and Sobol random number samplings in PSO for the mixed variable Cigar function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	Salp Swarm Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	93	97	3.1	6.8	N/A	0.47	0.41	4%
10	119	156	6.1	9.1	N/A	0.54	0.71	24%
15	147	209	11.2	11.8	N/A	0.73	0.87	30%
20	195	264	12.6	15.8	N/A	0.91	1.1	27%

Table 4. Comparison of performances of both Monte Carlo and Sobol random number samplings in PSO optimization for the continuous variable Paraboloid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	Salp Swarm (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	Salp Swarm Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	42	54	43	0.9	3.2	0.8	0.17	0.21	23%
10	67	91	71	1.2	4.5	0.9	0.26	0.37	26%
15	82	136	88	1.91	7.1	0.89	0.33	0.53	40%
20	102	178	120	1.95	7.9	1.1	0.42	0.59	43%

Table 5. Comparison of performances of both Monte Carlo and Sobol random number samplings in PSO for the mixed variable Paraboloid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	Salp Swarm (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	Salp Swarm Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	42	47	85	2.2	3.5	1.5	0.19	0.23	11%
10	62	68	98	3.5	4.9	1.69	0.27	0.38	20%
15	78	106	352	2.5	3.5	35.1	0.38	0.53	27%
20	94	151	364	4.5	8.2	30.2	0.46	0.64	38%

Table 6. Comparison of performances of both Monte Carlo and Sobol random number samplings in PSO for the continuous variable Ellipsoid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	Salp Swarm (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	Salp Swarm Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	48	60	44	2.2	3.7	0.875	0.20	0.24	20%
10	70	96	75	4.1	3.1	1.1	0.31	0.42	27%
15	92	154	96	3.9	5.1	2.8	0.41	0.68	40%
20	105	184	132	4.2	9.2	2.9	0.52	0.88	43%

Table 7. Comparison of performances of both Monte Carlo and Sobol random number samplings in PSO for the mixed variable Ellipsoid function.

Number of Dimensions	PSO Enhanced (Iteration)	PSO Conventional (Iterations)	Salp Swarm (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	Salp Swarm Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	45	53	90	3.5	4.6	11.9	0.26	0.26	15%
10	62	89	109	4.9	8.3	29.1	0.42	0.43	30%
15	84	128	342	4.4	6.7	54.3	0.52	0.65	35%
20	90	165	350	3.2	7.8	63.6	0.63	0.88	46%

Halton vs. Monte Carlo:

The results of Halton-based enhanced PSO are presented in Tables 8–13.

Table 8. Comparison of performances of both Monte Carlo and (scrambled) Halton random number samplings in PSO for the continuous variable Cigar function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	61	101	3.2	5.1	0.51	0.35	40%
10	111	189	5.1	6.3	0.77	0.64	42%
15	188	298	6.8	30.5	1.1	1.03	37%
20	220	451	10.1	30.1	1.3	1.51	52%

Table 9. Comparison of performances of both Monte Carlo and (scrambled) Halton random number samplings in PSO for the mixed variable Cigar function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	96	95	2.87	4.9	0.67	0.40	1%
10	107	156	4.12	5.8	0.81	0.65	32%
15	157	204	9.75	14.5	1.03	0.88	23%
20	190	265	13.3	8.31	1.30	1.16	29%

Table 10. Comparison of performances of both Monte Carlo and Halton random number samplings in PSO for the continuous variable Paraboloid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	38	52	0.9	3.1	0.41	0.20	30%
10	64	88	1.1	4.4	0.6	0.33	28%
15	79	142	1.15	6.5	0.76	0.52	45%
20	103	174	1.6	5.5	0.92	0.55	41%

Table 11. Comparison of performances of both Monte Carlo and Halton random number samplings in PSO for the mixed variable Paraboloid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	38	46	1.37	2.66	0.46	0.21	18%
10	60	77	3.1	4.31	0.62	0.33	23%
15	71	107	3.9	5.78	0.81	0.48	35%
20	92	146	6.99	7.52	0.94	0.67	38%

It can be seen from the above tables (Tables 2–13) that the quasi-random sequence-based enhanced PSO demonstrates superior performance compared to a conventional random number-based PSO for both continuous and mixed variable problems. The enhancement increases generally with a higher number of variables specifically for continuous variable problems. SOBOL works better for mixed variable functions than Halton, but Halton shows better convergence than SOBOL when considering most of the continuous variable benchmark problems.

Table 12. Comparison of performances of both Monte Carlo and Halton random number samplings in PSO for the continuous variable Ellipsoid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	38	60	2.7	3.7	0.6	0.24	40%
10	75	96	3.08	3.1	1.0	0.42	24%
15	86	154	3.01	5.1	1.27	0.68	45%
20	152	184	3.44	9.2	1.44	0.88	35%

Table 13. Comparison of performances of both Monte Carlo and Halton random number samplings in PSO for the mixed variable Ellipsoid function.

Number of Dimensions	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
5	42	53	2.54	4.6	0.50	0.26	21%
10	61	89	2.75	8.3	0.70	0.43	32%
15	110	128	6.19	6.7	0.91	0.65	14%
20	120	165	7.1	7.8	1.2	0.88	27%

Constant Inertia vs. Random Inertia:

As suggested earlier, in new variants proposed in the literature, adaptive parameter enhancements provide improvements. We can also replace the random numbers used in these variants with the quasi-random numbers suggested in our new algorithm as these two improvements are mutually exclusive. We illustrate this by comparing the constant inertia algorithm, which is the traditional approach to PSO, to the random inertia proposed as an enhancement [Tables 14–17].

Table 14. Comparison of random inertial weight-aided Sobol and conventional PSO performances for benchmark functions with 5 dimensions.

Function Type	PSO Sobol Contant Weight Inertia	PSO Conventional Random Weight Inertia	PSO Sobol Random Weight Inertia	Standard Deviation PSO Sobol Contant Weight Inertia	Standard Deviation PSO Conventional Random Weight Inertia	Standard Deviation PSO Sobol Random Weight Inertia	Improvement in Random Inertia (Conventional vs. Sobol)
Cigar—Continuous	85	54	48	1.9	1.88	3.9	11.1%
Ellipse—Continuous	48	35	30	2.2	1.2	1.8	14.3%
Parabola—Continuous	42	31	29	0.9	1.7	1.3	6.45%
Cigar—Mixed	93	108	95	3.1	14.3	3.6	12.03%
Ellipse—Mixed	45	57	47	3.5	8.5	4.6	17.54%
Parabola—Mixed	42	52	43	2.2	15.8	5.5	17.3%

In this section, we compare the performance of the PSO with the constant inertia weight and that of the PSO with the random inertia weight. Random inertia weight has been used in reference [20,21] to improve the iteration efficiency. The traditional constant inertia PSO, our proposed enhanced PSO with quasi-random numbers with constant inertia PSO, is compared with random inertia variants for both algorithms. We use Sobol random numbers for the enhanced PSO and its random inertia variant.

Table 15. Comparison of performances of random inertial weight-aided Sobol and conventional PSO for the benchmark functions with 10 dimensions.

Function Type	PSO Sobol Contant Weight Inertia	PSO Conventional Random Weight Inertia	PSO Sobol Random Weight Inertia	Standard Deviation PSO Sobol Contant Weight Inertia	Standard Deviation PSO Conventional Random Weight Inertia	Standard Deviation PSO Sobol Random Weight Inertia	Improvement in Random Inertia (Conventional vs. Sobol)
Cigar—Continuous	167	76	67	2.4	2.6	1.8	11.8%
Ellipse—Continuous	70	35	30	4.1	1.8	1.7	14.3%
Parabola—Continuous	67	31	29	1.2	2.3	1.6	6.5%
Cigar—Mixed	119	177	136	6.1	9.2	16.1	23.2%
Ellipse—Mixed	62	99	84	4.9	8.0	17.2	15.2%
Parabola—Mixed	62	89	71	3.5	8.2	4.5	20.2%

Table 16. Comparison of performances of random inertial weight-aided Sobol and conventional PSO for the benchmark functions with 15 dimensions.

Function Type	PSO Sobol Contant Weight Inertia	PSO Conventional Random Weight Inertia	PSO Sobol Random Weight Inertia	Standard Deviation PSO Sobol Contant Weight Inertia	Standard Deviation PSO Conventional Random Weight Inertia	Standard Deviation PSO Sobol Random Weight Inertia	Improvement in Random Inertia (Conventional vs. Sobol)
Cigar—Continuous	183	96	85	5.2	2.1	2.2	11.5%
Ellipse—Continuous	92	63	56	3.9	1.2	1.6	11.1%
Parabola—Continuous	82	61	52	1.91	1.3	1.9	14.8%
Cigar—Mixed	147	260	166	11.2	44.8	16.4	36.2%
Ellipse—Mixed	84	140	100	4.4	11.9	6.9	28.6%
Parabola—Mixed	78	133	86	2.5	30.1	5.7	35.3%

Table 17. Comparison of performances of random inertial weight-aided Sobol and conventional PSO for the benchmark functions with 20 dimensions.

Function Type	PSO Sobol Contant Weight Inertia	PSO Conventional Random Weight Inertia	PSO Sobol Random Weight Inertia	Standard Deviation PSO Sobol Contant Weight Inertia	Standard Deviation PSO Conventional Random Weight Inertia	Standard Deviation PSO Sobol Random Weight Inertia	Improvement in Random Inertia (Conventional vs. Sobol)
Cigar—Continuous	231	118	98	26.5	5.4	3.5	16.9%
Ellipse—Continuous	105	77	66	4.2	2.8	2.7	14.3%
Parabola—Continuous	102	74	62	1.95	2.1	1.9	16.2%
Cigar—Mixed	195	330	234	12.6	37.1	21.2	29.1%
Ellipse—Mixed	90	213	120	3.2	28.4	19.2	43.7%
Parabola—Mixed	94	193	105	4.5	23.6	5.1	45.6%

The velocity of each particle in the PSO algorithm at iteration n is updated according to the following equation:

$$v_{n+1}^{ij} = \omega * v_n^{ij} + c_1 * r_1 * \left(P_n^j - X_n^{ij}\right) + c_2 * r_2 * \left({G_n}^j - X_n^{ij}\right) \tag{16}$$

In the above equation, ω is the inertia weight. For constant inertia, ω was set to 0.75.

For random inertia weight, ω was set to $0.5 + \frac{rand}{2}$ as per [21]. The comparison results are provided in the table below. The improvement percentage in the table below is the comparison between random inertial weight-aided Sobol and conventional PSO variants.

From the above results, we conclude that the Sobol-based PSO variant performs better in terms of iteration efficiency with respect to the conventional PSO as well as random inertia PSO.

TSP Optimization Results

Tables 18 and 19 provide the TSP optimization comparisons using the PSO variants. TSP is a completely discrete problem, and SOBOL performs better, as can be seen from Figures 6–8, as well as Tables 14 and 15. For TSP, the particles were initialized using the standard uniformly distributed random numbers for both the standard and the enhanced PSOs. The velocity and position updates, which are basically a set of swaps, are controlled using the Sobol and Halton-generated random numbers in the case of enhanced PSO.

Table 18. Comparison of performances for Monte Carlo and Sobol random number samplings in PSO for TSP.

Number of Cities	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
10	74	103	16.1	28.9	0.98	1.18	28%
15	414	478	88.7	130.2	7.85	8.7	13%
20	1705	1927	210.2	371.4	16.55	17.8	11.5%

Table 19. Comparison of performances for Monte Carlo and Halton random number samplings in PSO for TSP.

Number of Cities	PSO Enhanced (Iterations)	PSO Conventional (Iterations)	PSO Enhanced Standard Deviation	PSO Conventional Standard Deviation	PSO Enhanced Compute Time (secs)	PSO Conventional Compute Time (secs)	Iterations' Improvement Percentage
10	78	103	18.2	28.9	1.26	1.18	24%
15	431	478	76.7	130.2	8.98	8.7	9.8%
20	1760	1927	212.1	371.4	17.39	17.8	8.6%

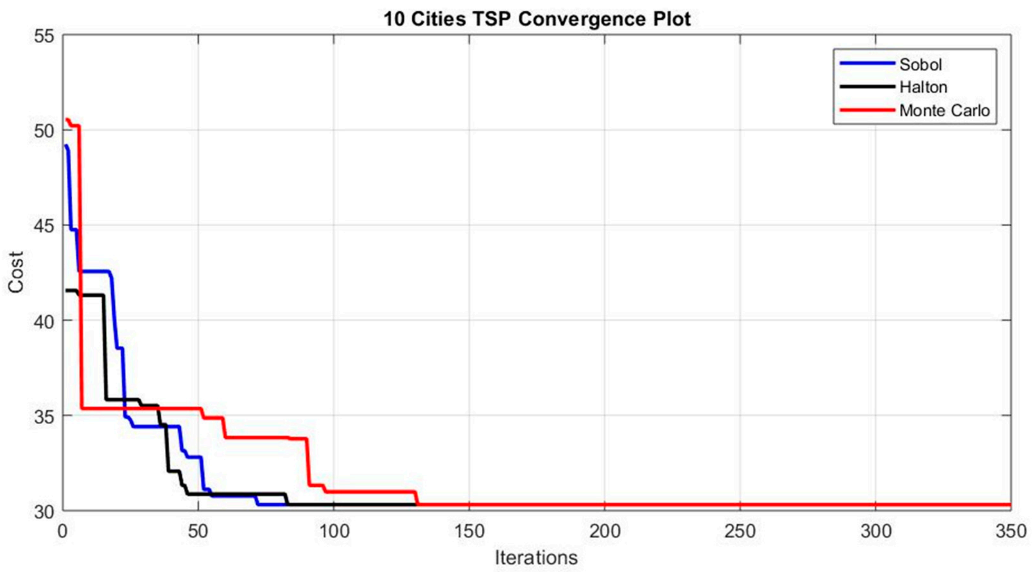

Figure 6. TSP with 10 cities—convergence plot.

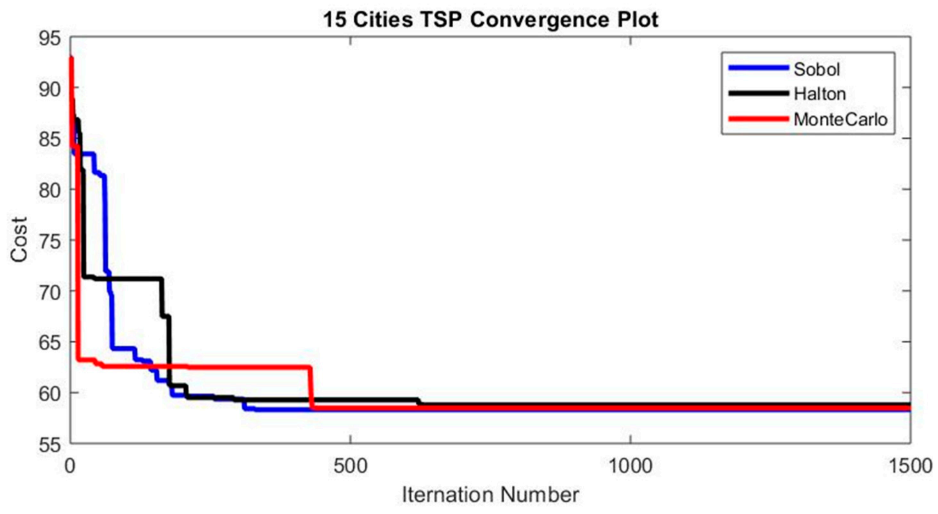

Figure 7. TSP with 15 cities—convergence plot.

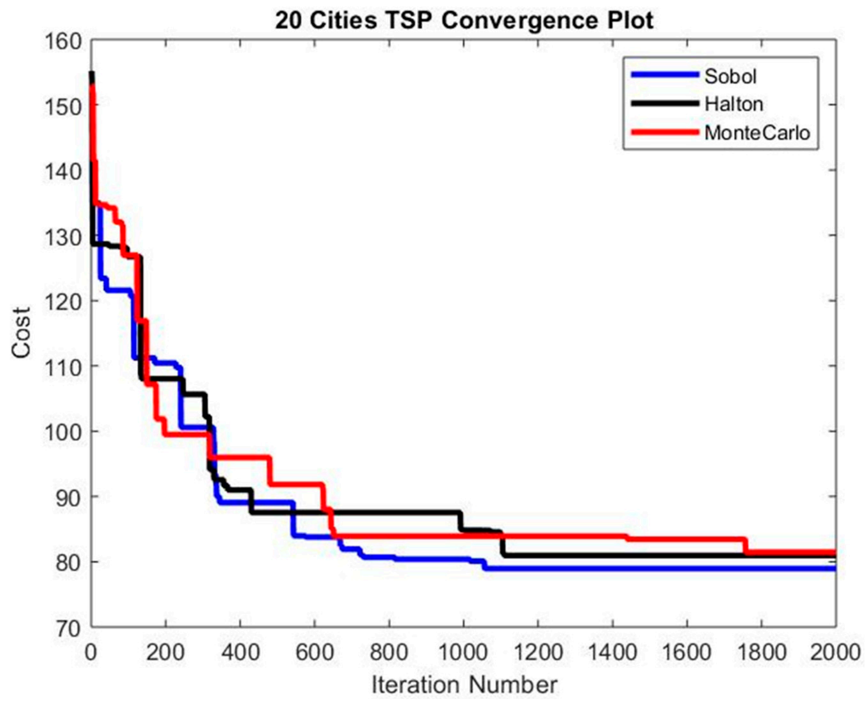

Figure 8. TSP with 20 cities—convergence plot.

4. Discussion

We have augmented the PSO algorithm by integrating Sobol and Halton random number samplings to achieve superior results. Our rationale behind this enhancement is rooted in Sobol and Halton random numbers, which are quasi-random number types. These numbers are generated in such a manner that they are uniformly distributed across multidimensional space, thus mitigating clustering or bias in particle movement. Consequently, this facilitates more extensive exploration, thereby increasing the likelihood of avoiding local optima and accelerating the discovery of the global optimum, as a more significant portion of the space can be explored when biases or clusters are circumvented.

The outcomes indicate that the enhancement applied to the standard PSO through the utilization of quasi-random numbers has consistently improved the number of iterations required for the algorithm to produce the optimal function value across all three benchmark functions. The efficiency gains for continuous functions are more pronounced than those for mixed variable functions.

Additionally, including more decision variables in the problem correlates with more significant improvements in general. As evident from the data, the application of Sobol or Halton sequences to the standard PSO algorithm demonstrates efficiency improvements, suggesting the potential benefits of these sequences to researchers in various optimization fields. Notably, this technique is algorithm-agnostic, as indicated in the introduction, and can thus be employed with other optimization algorithms such as the Genetic Algorithm, Ant Colony Optimization (ACO) algorithm, and other established optimization algorithms.

5. Conclusions

Particle Swarm Optimization harnesses swarm behavior effectively for function optimization but is often hampered by slow convergence. We consistently improved efficiency through two distinct enhancements to the Particle Swarm Optimization algorithm. We proposed the use of quasi-random number sequences to update decision variable values and their rates of change in each iteration to enhance the PSO algorithm. Since quasi-random number sequences do not alter the fundamental algorithm, this concept can be integrated into modified versions of the PSO algorithm suggested previously, as discussed in the introduction. To evaluate whether quasi-random number sequences in PSO yield efficiency improvements, we tested them using continuous and mixed variable Cigar, Ellipsoid, and Paraboloid functions, along with an example of the Traveling Salesman Problem. The results demonstrate that both sequences of quasi-random numbers used to enhance the standard PSO improved efficiency.

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

Funding: This research received no external funding.

Data Availability Statement: Data can be obtained from the authors.

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

References

1. 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]
2. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [CrossRef]
3. Bansal, J.C.; Singh, P.K.; Saraswat, M.; Verma, A.; Jadon, S.S.; Abraham, A. Inertia Weight strategies in Particle Swarm Optimization. In Proceedings of the 2011 Third World Congress on Nature and Biologically Inspired Computing, Salamanca, Spain, 19–21 October 2011; pp. 633–640. [CrossRef]
4. Li, N.; Qin, Y.-Q.; Sun, D.-B.; Zou, T. Particle swarm optimization with mutation operator. In Proceedings of the 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), Shanghai, China, 26–29 August 2004; Volume 4, pp. 2251–2256. [CrossRef]
5. Clerc, M.; Kennedy, J. The particle swarm–Explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.* **2002**, *6*, 58–73. [CrossRef]
6. Engelbrecht, A.P. Particle swarm optimization with crossover: A review and empirical analysis. *Artif. Intell. Rev.* **2016**, *45*, 131–165. [CrossRef]
7. Chen, Y.; Li, L.; Xiao, J.; Yang, Y.; Liang, J.; Li, T. Particle swarm optimizer with crossover operation. *Eng. Appl. Artif. Intell.* **2018**, *70*, 159–169. [CrossRef]
8. Zhou, Z.; Li, F.; Abawajy, J.H.; Gao, C. Improved PSO Algorithm Integrated with Opposition-Based Learning and Tentative Perception in Networked Data Centers. *IEEE Access* **2020**, *8*, 55872–55880. [CrossRef]
9. Mandal, B.; Roy, P.K.; Mandal, S. Hybridization of Particle Swarm Optimization with Biogeography-Based Optimization for Reactive Power and Voltage Control. In Proceedings of the 2014 Fourth International Conference of Emerging Applications of Information Technology, Kolkata, India, 19–21 December 2014; pp. 34–39. [CrossRef]

10. Cai, Y.; Yang, S.X. An improved PSO-based approach with dynamic parameter tuning for cooperative target searching of multi-robots. In Proceedings of the 2014 World Automation Congress (WAC), Waikoloa, HI, USA, 3–7 August 2014; pp. 616–621. [CrossRef]
11. Morokoff, W.J.; Caflisch, R.E. Quasi-Random Sequences and Their Discrepancies. *SIAM J. Sci. Comput.* **1994**, *15*, 1251–1279. [CrossRef]
12. Niederreiter, H. *Random Number Generation and Quasi-Monte Carlo Methods*; CBMS-NSF Regional Conference Series in Applied Mathematics 63; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 1992.
13. Sobol, I.M. Uniformly distributed sequences with an additional uniform property. *USSR J. Comput. Math. Math. Phys. Engl. Transl.* **1976**, *16*, 1332–1337. [CrossRef]
14. Diwekar, U.M.; Gebreslassie, B.H. Efficient Ant Colony Optimization (EACO) Algorithm for Deterministic Optimization. *Int. J. Swarm Intel. Evol. Comput.* **2016**, *5*, 131. [CrossRef]
15. Hadia, S.K.; Joshi, A.H.; Patel, C.K.; Kosta, Y.P. Solving City Routing Issue with Particle Swarm Optimization. *Int. J. Comput. Appl.* **2012**, *47*, 15.
16. Hadia, S.K.; Joshi, A.H.; Salman, C.A.; Ahmad, I.; Al-Madani, S. Particle swarm optimization for task assignment problem. *Microprocess. Microsyst.* **2002**, *26*, 363–371.
17. Zhong, W.H.; Zhang, J.; Chen, W.N. A novel discrete particle swarm optimization to solve traveling salesman problem. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Singapore, 25–28 September 2007; pp. 3283–3287.
18. Wang, K.P.; Huang, L.; Zhou, C.G.; Pang, W. Particle swarm optimization for solving traveling salesman problem. In Proceedings of the 2003 International Conference on Machine Learning and Cybernetic, Xi'an, China, 5 November 2003; Volume 3, pp. 1583–1585.
19. Mirjalili, S.; Gandomi, A.H.; Mirjalili, S.Z.; Saremi, S.; Faris, H.; Mirjalili, S.M. Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. *Adv. Eng. Softw.* **2017**, *114*, 163–191. [CrossRef]
20. Umapathy, P.; Venkataseshaiah, C.; Arumugam, M.S. Particle Swarm Optimization with Various Inertia Weight Variants for Optimal Power Flow Solution. *Discret. Dyn. Nat. Soc.* **2010**, *2010*, 462145. [CrossRef]
21. Nikabadi, A.; Ebadzadeh, M. Particle swarm optimization algorithms with adaptive Inertia Weight: A survey of the state of the art and a Novel method. *IEEE J. Evol. Comput.* **2008**, *14*, 305–313.
22. Ratnaweera, A.; Halgamuge, S.K.; Watson, H. Self-Organizing Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients. *IEEE Trans. Evol. Comput.* **2004**, *8*, 240–255. [CrossRef]
23. Zhan, Z.H.; Zhang, J.; Li, Y.; Chung, H.-H. Adaptive Particle Swarm Optimization. *IEEE Trans. Syst. Man Cybernetics. Part B Cybern.* **2009**, *39*, 1362–1381. [CrossRef] [PubMed]
24. Qu, B.Y.; Suganthan, P.N.; Das, S. A Distance-Based Locally Informed Particle Swarm Model for Multimodal Optimization. *IEEE Trans. Evol. Comput.* **2013**, *17*, 387–402. [CrossRef]
25. Shieh, H.-L.; Kuo, C.-C.; Chiang, C.-M. Modified particle swarm optimization algorithm with simulated annealing behavior and its numerical verification. *Appl. Math. Comput.* **2011**, *218*, 4365–4383. [CrossRef]
26. Feng, Z.-K.; Niu, W.-J.; Cheng, C.-T. Multi-objective quantum-behaved particle swarm optimization for economic environmental hydrothermal energy system scheduling. *Energy* **2017**, *131*, 165–178. [CrossRef]
27. Liu, D.; Xiao, Z.; Li, H.; Liu, D.; Hu, X.; Malik, O.P. Accurate Parameter Estimation of a Hydro-Turbine Regulation System Using Adaptive Fuzzy Particle Swarm Optimization. *Energies* **2019**, *12*, 3903. [CrossRef]
28. Wang, J.; Wang, X.; Li, X.; Yi, J. A Hybrid Particle Swarm Optimization Algorithm with Dynamic Adjustment of Inertia Weight Based on a New Feature Selection Method to Optimize SVM Parameters. *Entropy* **2023**, *25*, 531. [CrossRef] [PubMed]