Received October 23, 2020, accepted November 13, 2020, date of publication November 20, 2020, date of current version December 7, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3039602*

# Influence of Initializing Krill Herd Algorithm With Low-Discrepancy Sequences

**OVRE JEFFREY AGUSHAKA[1,2], (Member, IEEE), AND ABSALOM EL-SHAMIR EZUGWU[1], (Member, IEEE)**
[1]School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal-Pietermaritzburg Campus, Pietermaritzburg 3201, South Africa
[2]Department of Computer Science, Federal University of Lafia, Lafia 950101, Nigeria

Corresponding authors: Ovre Jeffrey Agushaka (218088307@stu.ukzn.ac.za) and Absalom El-Shamir Ezugwu (ezugwua@ukzn.ac.za)

This work was supported by the Tertiary Education Trust Fund under Grant TETF/ES/UNIV/NASARAWA//TSAS/2019.

**ABSTRACT** The krill herd (KH) algorithm is a global metaheuristic algorithm that was initially proposed for solving continuous optimization problems. The KH algorithm, since inception, has generated considerable real-world application interests in the research community. The standard algorithm solution implementation steps follow the initialization mechanism, which relies mainly on educated guesses or random initialization solution generation. Therefore, to improve the performance of the KH algorithm, the current study is set to investigate the influence of initializing the KH algorithm with three low-discrepancy sequences, such as the Faure sequence, Sobol sequence, and Van der Corput sequence. These low-discrepancy sequences are known to be more uniformly distributed across the problem search space than the commonly used random number initialization method. The study also evaluates the influence of population size on the performance of the proposed variants of the improved KH algorithms. The experimental results show significant improvements for the enhanced KH algorithms in terms of performance and the quality of solutions obtained; particularly on standard benchmarked high-dimension test problem instances, where the enhanced KH variants outperformed the existing basic KH algorithm for all the test functions evaluated. Similarly, the results for low dimension test cases showed less sensitivity to the initialization schemes, as the performance of our proposed improved scheme was comparable to that of the basic KH algorithm. However, in most cases, as the problem dimension was scaled up, the enhanced KH outperformed the basic KH. Evaluation results based on the population size of the algorithm, revealed that when the number of Krill is set at 25, the Sobol based KH initialization scheme performed better than did the other methods. Although, the Van der Corput and Faure based KH initialization schemes showed similar sensitivity when the dimension was set at 20. As we varied the population size of Krill, it was observed that the performance of the Sobol based KH initialization scheme deteriorated, whereas the other two methods showed superior performance. Overall, the findings from this study revealed that there are significant improvements in the performance of KH algorithm when initialized with low-discrepancy sequences.

**INDEX TERMS** Krill herd algorithm, initialization of metaheuristics, faure sequence, sobol sequence, van der caput sequence, low-discrepancy sequence.

## I. INTRODUCTION

In the krill herd algorithm, the dispersion of the initial Krill population in the search space contributes significantly to the algorithm's performance; if not dispersed effectively, the global optimum can be missed [1] through premature converging. A well-structured dispersion of random numbers can reduce this premature converging. Most random number generators use a uniform probability distribution to generate uniform pseudorandom number sequences; however, such a sequence does not have the lowest discrepancies [2]. The quasi-random number can be generated using a low-discrepancy sequence; these have been proven to have optimal discrepancy and are useful in optimization problems [3]. Low-discrepancy sequences, like those of Van der Corput, Sobol, Faure, and Halton, are potent computational method

The associate editor coordinating the review of this manuscript and approving it for publication was Corrado Mencar.

tools that have been used to improve optimization algorithms' performance.

The literature shows very few instances where the focus is on the application of quasi-random sequences initialization schemes to improve the performance of the metaheuristic algorithm. For example, the Van der Corput and Sobol sequences were shown to improve to the performance of the particle swarm optimization (PSO) when used to initialize the swarms [2]. In another study [4], the performance of the genetic algorithm (GA) was significantly improved when the Halton sequence was used to initialize the population of the genetic algorithm. Similarly, Uy *et al.* [5] conducted a comparative study of the use of Halton, Faure, and Sobol sequences for initializing the swarm population of the PSO algorithm. More so, previous studies have also shown that low-discrepancy sequences have greatly improved the performance of metaheuristic algorithms. More studies on the application of quasi-random sequences for initializing metaheuristic swarm populations can be found in [6]–[8]. The current study is set to investigate a similar influence of three quasi-random sequence initialization schemes on the krill herd algorithm (KH). These initialization schemes are abbreviated in this article as follows: Van der Corput (Vc), Sobol (So), and Faure (Fa). A brief discussion of the KH optimization algorithm is introduced next.

The KH metaheuristic optimization algorithm is inspired by the herding behavior of Krill [29]. The simplicity and ease of implementation of KH have caught the attention of many scholars resulting in many applications for and variants of the algorithm being published. More so, a detailed discussion of the KH algorithm is presented in section II of this article. The different forms of the KH algorithm have been generally classified into three main categories, which were proposed in [9]. The first classification covers the improved KH algorithms and includes chaotic KH [10], opposition KH [11], Lévy-flight KH [12], multi-stage KH [13] etc. The second classification covers the hybrid KH algorithms, and this includes the combination of cuckoo Search and KH [14], harmony search and KH [15], stud KH [16], biogeography-based KH [17], differential evolution based KH [18], etc. The third classification covers the variants of KH algorithms, and this includes discreet KH [19], binary KH [20], fuzzy KH [21], and multi-objective KH [22].

Many real-world applications of the KH algorithm exist in literature ranging from continuous optimization [23], combinatorial optimization [24], constrained optimization [25], multi-objective [26] and other related engineering domains [27]. Therefore, considering the relevance of the KH algorithm to the research communities, an effort is made in this article to enhance its performance by means of improving its initialization schemes. It is equally important to note here that the design of the original KH algorithm's initialization mechanism was based on the traditional random number generator scheme. This scheme has the limitation of clustering, thereby making the population spread to be non-uniform within the solution landscape. Further, traditional random number solution initialization schemes do not achieve the desired optimal discrepancy, which would aid the algorithm in achieving superior solution quality. These drawbacks motivate us to carry out our study on the three initialization methods mentioned above and to evaluate their influence on the KH algorithmic performance in finding a global optimum solution for difficult optimization problems.

To the best of our knowledge, no study has used a low-discrepancy sequence (particularly Van der Corput, Sobol or Faure) to improve the performance of KH. In this article, we investigate the effect of using Van der Corput, Sobol, and Faure sequences to initialize the krill population. The basic KH algorithm [29] will be used to test the influence of these sequences. We believe that our modification will have a ripple effect on all other variants of KH because they all require initialization of the krill population. The improved KH is then applied to the same benchmark functions used in [29] and the results compared with the basic KH. We also test the influence of the krill population and maximum iterations on the KH algorithms.

Based on the experimental investigations carried out in this article, we can highlight the features and technical contributions of this article as follows:

- The numerical experimentation results, which are based on the improvement of the KH algorithm by modifying its initialization method using three low-discrepancy sequences, namely VcKH, FaKH, and SoKH, show that under the same parameter condition of population size, the maximum number of iterations, and replications, the proposed enhanced KH algorithm variants perform better than standard KH algorithm. However, while the SoKH algorithm can easily a find near-optimal solution under a small population size and low problem dimension, the FaKH and VcKH performance is seen to increase under large population size and large problem dimension.
- The influence of the three initialization schemes on the standard KH algorithms is carefully evaluated in this article. The overall goal is to determine, in comparison with the basic KH algorithm, which of the three improved KHs has more sensitivity to initialization and is bound to producing better performance in terms of the optimal solution and computational efficiency.
- Comparative analyses of results of the three proposed initialization schemes are justified by using descriptive statistical values such as the mean result obtained after 20 algorithm run replications, standard deviation, mean error value, and the computational efficiency of each algorithm. We believe that the descriptive statistical results will be relevant when deciding on the appropriate initialization mechanisms to consider for any given optimization benchmark problem to solve using the KH algorithm or its variants.

The rest of the paper is organized as follows: a summary of related work, KH background details, and application areas are introduced in Section II of the paper. The study methodology, motivation, and the initialization schemes; namely, random number generation methods, Van der Corput sequence, Faure sequence, and Sobol sequence are introduced, then the proposed enhanced KH algorithmic design steps are given in Section III, Next, the experimental configuration, the test benchmark functions are outlined, and the results are discussed in Section IV. Finally, the conclusion and future research direction are given in section V.

## II. RELATED WORK

In most metaheuristics, optimization algorithms initialization of the swarm or individual population usually involves some randomness or informed guesses, and these have considerable influence on the algorithms' ability to find optimal solutions. However, linear programs and convex optimizers attain optimal solutions independent of the initialization schemes. The fact is most optimization algorithms are nonlinear and non-convex [50]. Different mathematical and statistical approaches have been previously proposed to solve the initialization problem.

The idea of uniformly covering the search space in order to obtain the optimal solution is well researched. Studies have shown that quasi-random numbers are efficient initialization mechanisms for metaheuristic algorithms searching for near-optimal solutions. For example, the swarm population in [5] was initialized using Halton, Sobol, and Faure randomized low-discrepancy sequences, and the results were then compared with those from the global best PSO. While there was a significant improvement in the PSO with Sobol, there were mixed results for the Faure and Halton sequences. The authors in [5], therefore, concluded that employing low-discrepancy sequences for the initialization scheme of metaheuristic algorithms significantly improves the quality of solutions obtained by the algorithms. Similarly, Pant *et al.* [2], used the Van der Corput and Sobol sequences to initialize the swarm population of the PSO. They compared their result with those from the basic PSO, which uses a uniform distribution mechanism for the initialization of the PSO swarm. Their results showed that the improved PSO with a low-discrepancy sequence yielded a better initialized swarm and increased the performance of PSO by a significant percentage.

Quasi-random numbers, especially those with low discrepancies, have shown promising results; however, severe limitations exist in their usage. For instance, they perform poorly as the dimension of the problem gets bigger [51]. However, the advantages that they present are a great asset to their usage by researchers.

Brits *et al.* [7] used niching methods to initialize the population of a swarm to find multiple solutions to function optimization problems. Using the guaranteed convergence particle swarm optimization (GCPSO) algorithm, their results showed that all maxima were successfully located for all the simulation runs. However, the GSPSO had some drawbacks, such as its poor performance on high dimensional functions and high computational cost. The swarms in [30] were initialized using a nonlinear simplex method, and the results showed that the particles gravitated better toward the good quality solutions than they had with the variant of PSO considered in the paper.

A different approach was considered by Richards and Ventura in [31], where one particle was placed in the centre, and the remainder were spread around it in the search space. Their result was promising; however, it is not entirely without bias. The chaos-based approach involves generating random numbers using a chaotic map and fewer initial conditions. This approach is dependent on initial conditions [52].

Researchers have also designed schemes in conjunction with other algorithms to solve problem-specific initialization problems. For instance, Kondamadugula and Naidu [50] combined a special sampling evolutionary algorithm with a random sampling evolutionary algorithm to tune the parameters in a digital integrated circuit. In Li *et al.* [53], the GA was initialized using a knowledge-based system to solve the traveling salesman problem. The degrees of the node for each neighbourhood was evaluated in [54], and based on these degrees, the population was initialized to solve the network disintegration problem. However, this approach has setbacks such as increasing the computational cost, it is problem-specific, and its performance is greatly influenced by the expertise of the user.

The authors in [55] carried out a systematic comparison of the effect of 22 different probability distribution initialization methods on the convergence and accuracy of five optimization algorithms. Their results showed that the population size and maximum number of iterations affect most algorithms differently. It also showed that some algorithms are insensitive to the initialization scheme, and overall, initialization of the population plays a significant role in finding an optimal solution for some problems.

Next, we present a preliminary discussion on the KH algorithm, which includes both KH background and major application areas from literature.

### A. KRILL HERD ALGORITHM

The KH algorithm was first proposed by Gandomi and Alavi in 2012 [29], and the algorithmic inspiration is based on the simulation of the herding behavior of krill individuals. The algorithmic procedure for the KH is as given in algorithm listing 1, while a schematic representation of the sensing ambit around a krill individual is shown in Figure 1. The objective of the algorithm is to minimize the distance of each krill from food location and krill density. Three basic motions are defined for each krill; specifically, the motion induced by other krill, foraging motion, and physical diffusion. The algorithm starts by initializing the krill position using a random number generator (or as in our proposed improvement, a low-discrepancy sequence generator, as is discussed in this article).

For each krill, the three aforementioned motions are evaluated, the scale factor is determined, and the value of the fitness

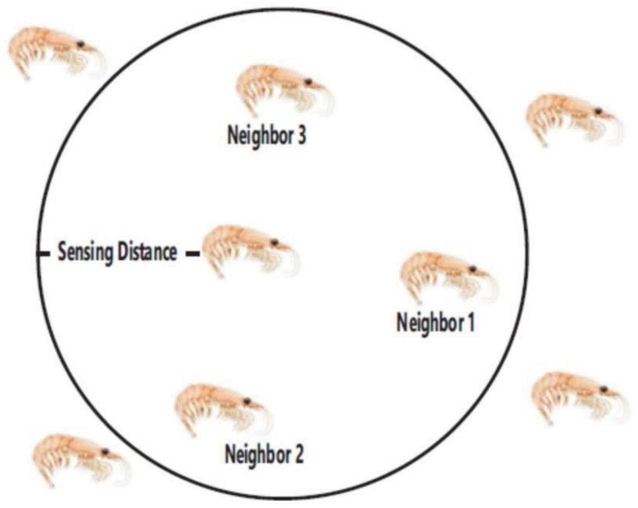


**FIGURE 1.** An illustration of the sensing ambit between the individual krill [29].

function for each krill is calculated as follows:

$$\frac{dX_i}{dt} = (N_i + F_i + D_i) \tag{1}$$

where $N_i$ is the motion induced by other krill, $F_i$ is the foraging motion and $D_i$ is the physical diffusion. To improve the performance of the algorithm, some genetic reproduction mechanisms such as crossover and mutation can be incorporated into the KH algorithm. For the KH algorithm, the movement induced by other krill is defined as follows:

$$N_i^{new} = N^{max}\alpha_i + \omega_i N_i^{old} \tag{2}$$

where $\alpha_i$ is the direction of motion, which is influenced by the target krill density, local krill density, and repulsive krill density. The foraging motion is influenced by the food location and previous knowledge of the location of the food, and this is defined as

$$F_i = V_f\beta_i + \omega_f F_i^{old} \tag{3}$$

where

$$\beta_i = \beta_i^{food} + \beta_i^{best}. \tag{4}$$

the parameter $V_f$ is the foraging speed, $\omega_f$ is the inertia weight and $F_i^{old}$ is the last foraging movement. The physical diffusion is defined in terms of maximum diffusion speed ($D^{max}$) and a directional vector ($\delta$), which is defined as follows

$$D_i = D^{max}\delta. \tag{5}$$

There is a need for the physical diffusion to decrease with time, so equation (5) is redefined as

$$D_i = D^{max}(1 - \frac{I}{I_{max}})\delta. \tag{6}$$

The individual krill position vector at time $t$ to $t + \Delta t$ is given as

$$X_i(t + \Delta t) = X_i(t) + \Delta t\frac{dX_i}{dt}. \tag{7}$$

**Algorithm 1** Standard Krill Herd Algorithm [48]

```
1: $k \leftarrow 1$ {initialization}
2: Initialize parameters ($Dmax$, $N\ max$, etc.)
3: **For** $i = 1$ to $M$ **do**
4: Generate Solution ($xi(k)$)
5: {evaluate and update best solutions}
6: $K(xi(0)) \leftarrow$ Evaluate quality($xi(0)$)
7: **End for**
8: $x* \leftarrow$ Save best individual $x*(0)$
9: {main loop}
10: **Repeat**
11: sort population of krills
12: **For** $i = 1$ to $M$ **do**
13: Perform motion calculation and genetic operators:
14: $Ni \leftarrow$ Motion induced by other individuals
15: $Fi \leftarrow$ Foraging activity
16: $Di \leftarrow$ Random diffusion
17: Crossover
18: Mutation
19: {update krill position}
20: Update Solution ($xm(k)$)
21: {evaluate and update best solutions}
22: $K(xi(k)) \leftarrow$ Evaluate quality($xi(k)$)
23: **End for**
24: $x* \leftarrow$ Save best individual $x*(k)$
25: stop condition $\leftarrow$ Check stop condition ()
26: $k \leftarrow k + 1$
27: **Until** stop condition $=$ **False**
28: **Return** $K(x*(k))$, $x*(k)$, $k$
```

$\Delta t$ is the scale factor, which is tuned according to the optimization problem.

Next, the krill positions are updated based on the motions evaluated earlier, and the iteration continues until a global value is attained or the maximum number of iterations is reached. Algorithm listing 1 shows the implementation steps of the standard krill herd algorithm design. See [29] for a comprehensive discussion on the KH algorithm.

Since it was first published in 2012, the KH has received significant attention from researchers, and according to Google Scholar, the original publication had been cited 1177 times by August 2020. A search in Google Scholar, IEEE, Scopus, and Web of Science, in conjunction with [9] showed that a total of 122 articles related to KH had been published as of August 2020. In August 2020, there were already 20 articles published for the year. Figure 2 shows the distribution according to years of publication.

## B. APPLICATION AREAS OF KH

The 122 articles published related to KH show its application in many areas, which can be classified into continuous optimization, combinatorial optimization, constrained optimization, multi-objective optimization, dynamic, and noisy environment engineering, and fuzzy systems. Table 1 gives

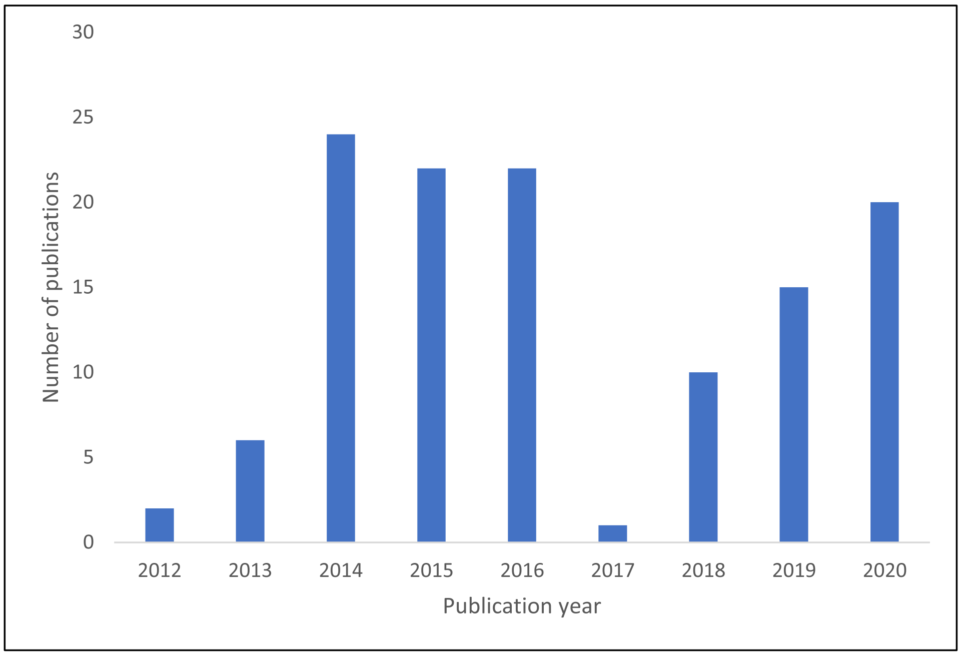


**FIGURE 2.** Number of KH related publications.

a summary of these application areas and the number of publications in each [9], [56]–[69].

The clustered column chart, presented in Figure 3, visually summarizes and compares the number of publications across the various application areas identified in Table 1. We see that the optimal power flow problem is the area with the most applications of the KH algorithm. It also shows that the KH algorithm is applied widely; there are more than ten application areas in which the algorithm has more than five publications. These span promising areas of practical optimization problems.

## III. METHODOLOGY

The performance of population-based metaheuristic algorithms is heavily dependent on the initialization of the swarms and the initial solution generated in this process. With little or no knowledge about the search or solution space, the spread of these swarms over the space is crucial [34]. The most common method of initialization is the use of randomly-generated initial solutions based on the concept of a random number generator mechanism. However, there are some major limitations associated with the traditional random initialization schemes and so using low-discrepancy sequences is a viable alternative for the process. In this section, we discuss each of these methods of initialization and highlight their strengths and weaknesses.

### A. MOTIVATION

The role that initialization plays in the final solution for most optimization algorithms is uncontested. Several initialization schemes exist, as discussed in the previous section, all with their advantages and disadvantages. The spread of the initial population over the search space is important, and different initialization distributions schemes do this differently. Our choice of low-discrepancy sequences hinged on their offering a uniform and wide distribution in the search space, which different research perspectives have shown to, respectively, influence the initial and the final results [2], [5]. The ease of use and implementation of these initialization mechanisms is another factor that influenced our choice for the current study. Although some of the low-discrepancy sequences initialization schemes perform poorly as the problem dimension or graph size scales up, their advantages are of still significant, especially when compared to the well-known and commonly-used random number initialization method. Overall, just as other initialization schemes have their advantages and disadvantages, it is a trade-off between computational cost and solution quality.

While researchers have done great work in applying these schemes on algorithms such as PSO, GA, ABC, to the best of our knowledge, no study has been published on their use in the initialization of KH. This reason alone would be sufficient motivation for the current study. Further, we were also motivated by our interest in finding whether the KH algorithm would be sensitive to the choice of the initialization scheme, particularly the low-discrepancy sequences.

Next, we briefly introduce the four initialization schemes that were earlier mentioned in this text.

### B. RANDOM NUMBER GENERATION OR MONTE CARLO METHODS

The random number generation or Monte Carlo methods play an essential role in the initialization process [35], [36]. The main point here is how genuinely random are the resulting

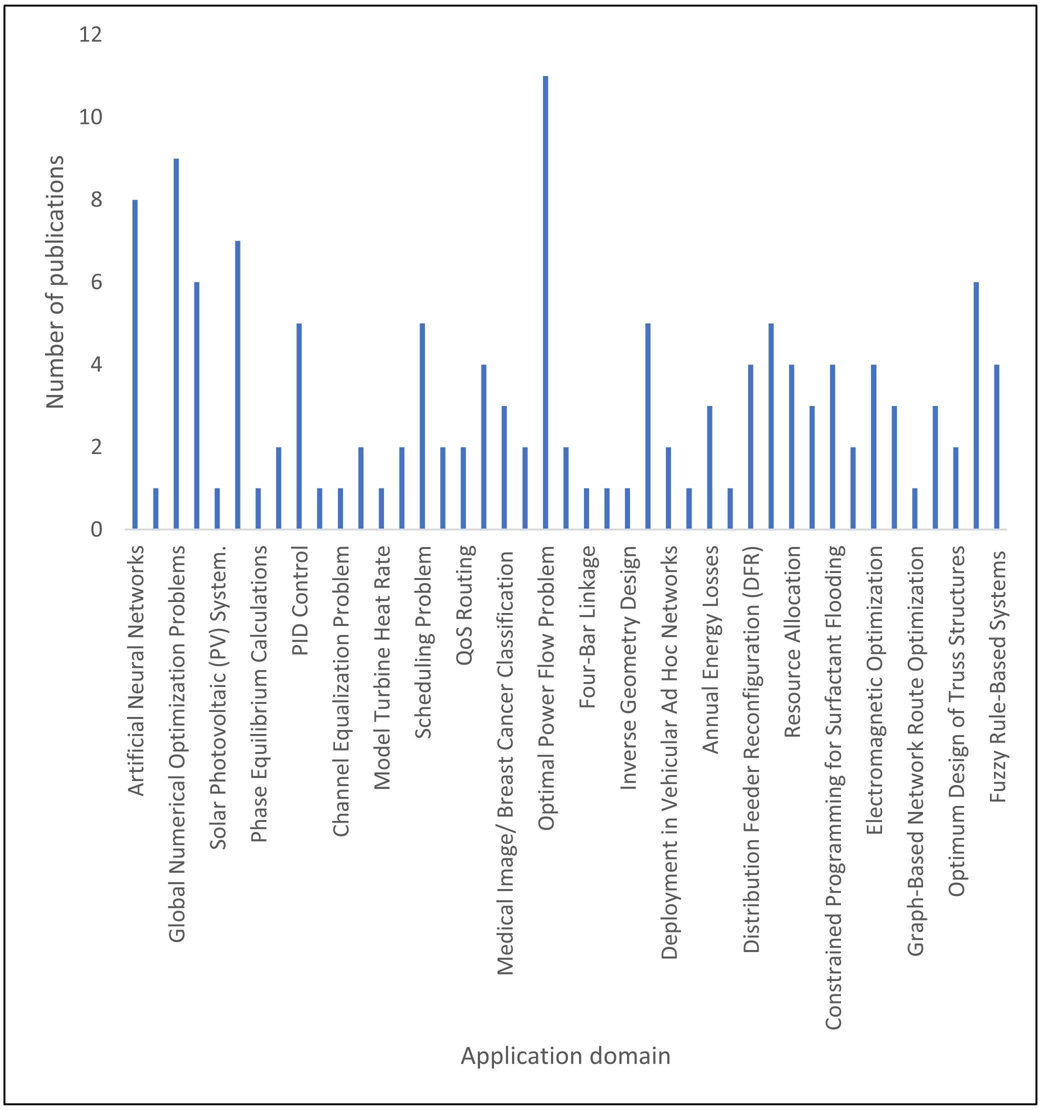


**FIGURE 3.** Clustered chart for application areas of KH.

'randomly generated' solutions within the solution search spaces. The quality of a pseudorandom generator is dependent on its measure of discrepancy [32]. We illustrate this further as follows:

Given a set of points $P = \{X_1, X_2, \ldots, X_N\} \in [0 \ldots 1]^s$ the star discrepancy of P is given in [37] as

$$T_N^*(P_N) = \sqrt{\int_{[0,1]^s}[\frac{A(J, P_N)}{N} - V(J)]^2 du} \tag{8}$$

where $J = [0..u_i]\ \forall u = (u_1, u_2, \ldots, u_s), A(J, P_N)$ is the number of points in $J$ and $V(J)$ is the volume. The linear congruential method is one of the most widely used random number generations, it uses the Lehmer sequence, and it is of the form [38]

$$x_i = (ax_i + c)\, mod\ m \quad \forall 0 \le x_i \le m \tag{9}$$

The choice of $a$ and $m$ has a significant influence on the random numbers generated, and normally $c$ is set to 0. It has been shown in [37], [39] that Monte Carlo methods do not achieve optimal discrepancy that would aid the algorithm to achieve superior quality results. These limitations motivated us to drop the Monte Carlo methods for quasi-random methods with low discrepancies. However, other initialization schemes such as those based on probability distributions exist in literature. The descriptions of the three quasi-random methods adopted for our work are given in the next sections.

### C. VAN DER CORPUT SEQUENCE

The Van der Corput sequence is the basis for most low-discrepancy sequences. It was initially defined for one-dimensional space and base $(b \geq 2)$ [28]. It is defined as follows. Given that $\varphi_b = N_0 \rightarrow [0, 1)$, if $n \in N_0$ then $n$ can be expanded as

$$n = \sum_{j=0}^{T} a_j b^{j-1} \tag{10}$$

**TABLE 1.** Application area of KH.

| Category | Area of Application | Number of Publications |
|---|---|---|
| Continuous Optimization [9, 56, 70] | Artificial Neural Networks | 8 |
| | Daily Pan Evaporation-Climatology | 1 |
| | Global Numerical Optimization Problems | 9 |
| | Optimizing Support Vector Regression/ Machine | 6 |
| | Solar Photovoltaic (PV) System. | 1 |
| | Clustering Problem | 7 |
| | Phase Equilibrium Calculations | 1 |
| | Sensorless Control | 2 |
| | PID Control | 5 |
| | Inverse Radiation Problem | 1 |
| | Channel Equalization Problem | 1 |
| | Wireless Sensor Networks | 2 |
| | Model Turbine Heat Rate | 1 |
| Combinatorial Optimization [19, 70] | Flexible Job-Shop Scheduling Problem (FJSSP) | 2 |
| | Scheduling Problem | 5 |
| | Anomaly Detection Algorithm | 2 |
| | QoS Routing | 2 |
| | Portfolio Optimization | 4 |
| | Medical Image/ Breast Cancer Classification | 3 |
| | Feature Selection | 2 |
| | Optimal Power Flow Problem | 11 |
| | Mobility Tracking Problem | 2 |
| | Four-Bar Linkage | 1 |
| | Optimal Capacitor Allocation Problem | 1 |
| | Inverse Geometry Design | 1 |
| Constrained Optimization [70] | Economic Dispatch Problem | 5 |
| | Deployment in Vehicular Ad Hoc Networks | 2 |
| | CHPED Problem | 1 |
| | Annual Energy Losses | 3 |
| | Distributed Generator (DG) | 1 |
| | Distribution Feeder Reconfiguration (DFR) | 4 |
| | Optimal Reactive Power Dispatch (ORPD) | 5 |
| | Resource Allocation | 4 |
| | Optimal VAR Dispatch Problem | 3 |
| | Constrained Programming for Surfactant Flooding | 4 |
| | Transient Stability Constrained Optimal Power Flow (TSCOPF) | 2 |
| Multiobjective Optimization [22,26] | Electromagnetic Optimization | 4 |
| | Grid Fault Recovery Assistant Decision | 3 |
| Dynamic and Noisy Environment [6, 46] | Graph-Based Network Route Optimization | 1 |
| Engineering [70] | Structural Optimization | 3 |
| | Optimum Design of Truss Structures | 2 |
| | Engineering Optimization Problems | 6 |
| Fuzzy System [21, 30] | Fuzzy Rule-Based Systems | 4 |

where $a_j \in \{0, \ldots, b-1\}$, and $T = \lfloor log_b n \rfloor .\varphi_b$, and the parameter $\varphi_b$ can be formally defined as

$$\varphi_b(n) = \sum_{j=0}^{T} \frac{a_j}{b^{j+1}} \tag{11}$$

where $\varphi_b(n)_{n\geq 0}$ is the Van der Corput sequence with base $b$. Once defined, the sequence members are densely populated around the unit interval. Figure 4 shows how the Van der Corput sequence is distributed as compared to the distribution according to a pseudorandom number scheme. In our paper, we use the Van der Corput to generate a low-discrepancy sequence to initialize the krill positions, forming a dense population around the search space of the problem [49].

## D. FAURE SEQUENCE

The Halton sequence [32] is one of Van der Corput's extensions, and the Faure sequence [33] is a permutation of the Halton sequence. The Faure sequence is known to be better distributed uniformly in the search space when compared to that of Sobol (see Figure 5). It is defined as follows.

$$\text{Given } Z_k \equiv (C_1, C_2, \ldots, C_d) \tag{12}$$

where $C_i$ is the Halton sequence. The parameter $m$ is the smallest prime number greater than or equal to the dimension of the problem and not less than 2. The Faure sequence is generated as follows

$$\text{If } C_n = b_0^{m-1} + b_1^{m-2} + \ldots + b_r^{m-(r+1)} \tag{13}$$

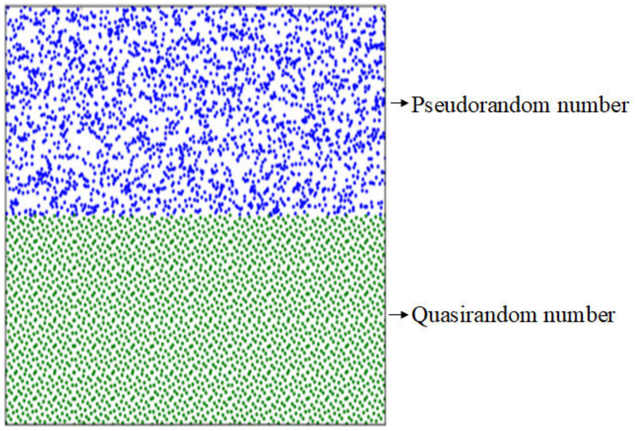


**FIGURE 4.** **Pseudorandom sequence vs. Van der Corput sequence.**

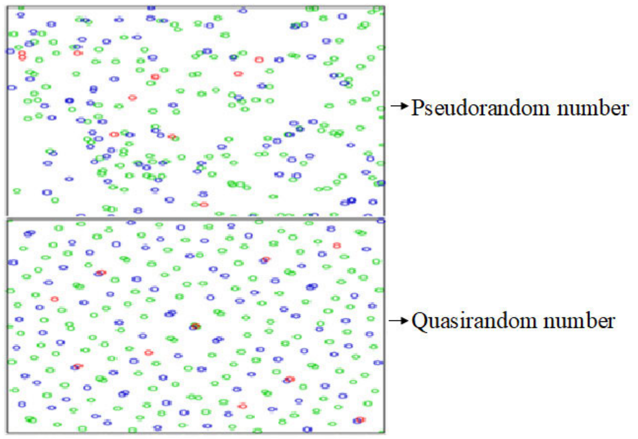


**FIGURE 5.** **Pseudorandom vs. Faure population spread.**

then

$$C_{n-1} = a_0^{m-1} + a_1^{m-2} + \ldots + a_r^{m-(r+1)} \tag{14}$$

where

$$b_j \equiv \sum\nolimits_{i \geq j}^{r} \binom{i}{j} a_i \; mod \; m. \tag{15}$$

The Halton and Faure sequences are known to perform poorly when the search space has large dimensions. However, the distribution around the search space is satisfactory, as similarly highlighted in [2]. In Figure 5, we demonstrate, through a simulation run, the population spread or distribution topology of the Pseudorandom and Faure sequences.

## E. SOBOL SEQUENCE

A Sobol function $(X_n^{(j)})$, generates sequences faster computationally than the Faure method. It is defined as follows [40]–[42].

Given $F_2 = \{0 \; 1\}$ and a linear nonrecurrence relation defined over it, $n > 0$ can be expanded as

$$n = n_1 2^0 + n_2 2^1 + \cdots + n_w 2^{w-1} \tag{16}$$

$$X_n^{(j)} = n_1 v_1^{(j)} \oplus n_2 v_2^{(j)} \oplus \ldots \oplus n_w v_w^{(j)} \tag{17}$$

where

$$v_i^{(j)} = a_1 v_{i-1}^{(j)} \oplus a_2 v_{i-2}^{(j)} \oplus \ldots \oplus a_q v_{i-q+1}^{(j)} \oplus v_{i-q}^{(j)} \oplus \ldots \oplus (v_{i-q}^{(j)}/2^q) \tag{18}$$

$i > q$, $a_i$ is the coefficient of the $qth$ primitive polynomial of $F_2$ and this is used to generate the Sobol sequence. Figure 6 shows the sample simulation output, indicating how the Sobol sequence is distributed.

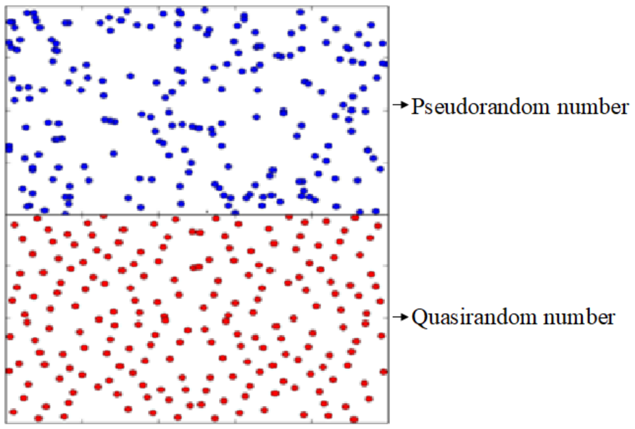


**FIGURE 6.** **Pseudorandom vs. Sobol distributions.**

## F. PROPOSED ALGORITHM

The population of krill in the basic KH algorithm [29] are initialized using the pseudorandom number generator, which has been proven to have sub-optimal discrepancy [43]. With the advantages of low-discrepancy sequences like the Van der Corput, Sobol, and Faure sequences, we evaluate the effects of these sequences on the basic KH initialization.

### 1) VAN DER CORPUT SEQUENCE-BASED KH ALGORITHM

Here, we initialized the krill population using the Van der Corput sequence (equation 11). A function that generates the Van der Corput sequences was implemented in MATLAB. The random number generator used in the basic KH is replaced with this function, which generates the Van der Corput sequence at every iteration of the algorithm, thereby providing a better spread of the krill over the solution search space. This spread then increases the probability of convergence to the optimal solution. The steps of the Van der Corput KH (VcKH) algorithm are depicted in algorithm listing 2.

The Van der Corput function, vdcorput $(k, b)$, takes two parameters; $k$ is the maximum sequence index, which is a non-negative integer, and $b$ is the sequence base integer exceeding 1. The output: s, is a $(k + 1) * 1$ array, with $s(i)$ storing $(i + 1)$ Van der Corput sequences.

### 2) FAURE SEQUENCE-BASED KH ALGORITHM

Using equation 14, the Faure function that generates the Faure sequences is used in place of the random number

**TABLE 2. Benchmark functions.**

| ID | Name | Dimension | Function | Bounds | Global |
|---|---|---|---|---|---|
| **F1** | Ackley | 30 | $f(x) = -a\,exp\left(-0.02\sqrt{n^{-1}\sum_{i=1}^{n} x_i^2}\right) - exp\left(n^{-1}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + a + e, a = 20$ | $X\epsilon[-32,32]^{30}$ | 0 |
| **F2** | Griewank | 30 | $f(X) = 1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right)$ | $X\epsilon[-600,600]^{30}$ | 0 |
| **F3** | Quartic | 30 | $f(X) = \sum_{i=1}^{n} ix_1^4 + rand$ | $X\epsilon[-1.28,1.28]^{30}$ | 0 |
| **F4** | Rastrigin | 30 | $f(X) = 10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$ | $X\epsilon[-5.12,5.12]^{30}$ | 0 |
| **F5** | Rosenbrock | 30 | $f(X) = \sum_{i=1}^{n}(100(x_i - x_i^2)^2 + (x_i - 1)^2)$ | $X\epsilon[-30,30]^{30}$ | 0 |
| **F6** | Schwefel 2.26 | 30 | $f(X) = \sum_{i=1}^{n} -x_i sin\left(\sqrt{\lvert x_i\rvert}\right)$ | $X\epsilon[-500,500]^{30}$ | 0 |
| **F7** | Schwefel 2.22 | 30 | $f(X) = \sum_{i=1}^{n}\lvert x_i\rvert + \prod_{i=1}^{n}\lvert x_i\rvert$ | $X \in [-100,100]^{30}$ | 0 |
| **F8** | Michalewicz | 10 | $-\sum_{i=1}^{d}\sin(x_i)\,sin^{2m}\left(\frac{ix_i^2}{\pi}\right)$ | $X \in [0,\pi]$ | -0.966015 At d=10 |
| **F9** | Schwefel 1.2 | 30 | $f(X) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$ | $X \in [-100,100]^{30}$ | 0 |
| **F10** | Sphere | 30 | $f(X) = \lVert X\rVert, \lVert X\rVert = \sqrt{\sum_{i=1}^{n} x_i^2}$ | $X\epsilon[-100,100]^{30}$ | 0 |
| **F11** | Booth | 2 | $f(X) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$ | $X \in [-10,10]^2$ | 0 |
| **F12** | Hartman 4-Dimensional | 4 | $f(X) = \frac{1}{0.839}\left[1.1 - \sum_{i=1}^{4}\alpha_i exp\left(-\sum_{j=1}^{4} A_{ij}(x_j - P_{ij})^2\right)\right]$ $\alpha = (1.0, 1.2, 3.0, 3.2)'$ $A = \begin{pmatrix} 10,3,17,3.5,1.7,8 \\ 0.05,10,17,0.1,8,14 \\ 3,3.5,1.7,10,17,8 \\ 17,8,0.05,10,0.1,14 \end{pmatrix}$ $P = 10^{-4}\begin{pmatrix} 1312,1696,5569,124,8283,5886 \\ 2329,4135,8307,3736,1004,9991 \\ 2348,1451,3522,2883,3047,6650 \\ 4047,8828,8732,5743,1091,381 \end{pmatrix}$ | $X \in [0,1]$ | -3.86278 |
| **F13** | Hartmann 6-dimensional | 6 | $f(X) = -\sum_{i=1}^{4}\alpha_i exp\left(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij})^2\right)$ $\alpha = (1.0, 1.2, 3.0, 3.2)'$ $A = \begin{pmatrix} 10,3,17,3.5,1.7,8 \\ 0.05,10,17,0.1,8,14 \\ 3,3.5,1.7,10,17,8 \\ 17,8,0.05,10,0.1,14 \end{pmatrix}$ $P = 10^{-4}\begin{pmatrix} 1312,1696,5569,124,8283,5886 \\ 2329,4135,8307,3736,1004,9991 \\ 2348,1451,3522,2883,3047,6650 \\ 4047,8828,8732,5743,1091,381 \end{pmatrix}$ | $X \in [0,1]$ | -3.32237 |

TABLE 2. *(Continued.)* Benchmark functions.

| | | | | | |
|---|---|---|---|---|---|
| **F14** | Hartmann 3-dimensional | 3 | $f(X) = -\sum_{i=1}^{4} \alpha_i exp\left(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2\right)$ $\alpha = (1.0, 1.2, 3.0, 3.2)'$ $A = \begin{pmatrix} 3.0, 10, 30 \\ 0.1, 10, 35 \\ 3.0, 10, 30 \\ 0.1, 10, 35 \end{pmatrix}$ $P = 10^{-4}\begin{pmatrix} 3689, 1170, 2673 \\ 4699, 4387, 7470 \\ 1091, 8732, 5547 \\ 381, 5743, 8828 \end{pmatrix}$ | $X \in [0,1]$ | -3.86278 |
| **F15** | Shekel | 4 | $f(X) = -\sum_{i=1}^{m}\left(\sum_{j=1}^{4}(x_j - C_{ji})^2 + \beta_i\right)^{-1}$ $m = 10$ $\beta = \frac{1}{10}(1,2,2,4,4,6,3,7,5,5)^T$ $\begin{pmatrix} 4.0\ 1.0\ 8.0\ 6.0\ 3.0\ 2.0\ 5.0\ 8.0\ 6.0\ 7.0 \\ 4.0\ 1.0\ 8.0\ 6.0\ 7.0\ 9.0\ 3.0\ 1.0\ 2.0\ 3.6 \\ 4.0\ 1.0\ 8.0\ 6.0\ 3.0\ 2.0\ 5.0\ 8.0\ 6.0\ 7.0 \\ 4.0\ 1.0\ 8.0\ 6.0\ 7.0\ 9.0\ 3.0\ 1.0\ 2.0\ 3.6 \end{pmatrix}$ | $X \in [0,1]$ | -10.5364 |
| **F16** | Zakharov | 30 | $f(X) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^2 + \left(\sum_{i=1}^{d} 0.5ix_i\right)^4$ | $X \in [-5,10]$ | 0 |
| **F17** | Easom | 2 | $f(X) = -\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2 - (x_2-\pi)^2)$ | $X \in [-100,100]$ | -1 |
| **F18** | Branin | 2 | $f(X) = a(x_2 - bx_1{}^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s$ | $x_1 \in [-5,10]$ $x_2 \in [0,15]$ | 0.397887 |
| **F19** | Colville | 4 | $f(X) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3{}^2 - x_4)^2 + 10.1(x_2 - 1)^2 + (x_4 - 1)^2 + 19.8(x_2 - 1)(x_4 - 1)$ | $X \in [-10,10]$ | 0 |
| **F20** | Goldstein-Price | 2 | $f(X) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1{}^2 - 14x_2 + 6x_1x_2 + 3x_2{}^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1{}^2 + 48x_2 - 36x_1x_2 + 27x_2{}^2]$ | $X \in [-2,2]$ | 3 |

**Algorithm 2** Pseudocode for the Van der Corput Sequence Improved KH Algorithm

```
Begin
Initialization. Set generation counter; initialize population $NP$ krill; set the foraging speed $V_f$, maximum diffusion speed $D_{max}$, and maximum induced speed $N_{max}$; probability of crossover $pc$.
Evaluate population. Evaluate the krill population based on its position.
For dimension d
    Krill position = (between upper bound and lower bound). * vdCorput( )
 End
Krill = evaluate (krill position)
While MaxGeneration do
        Sort all the Krill according to their fitness.
        For all Krill do
            Perform the three motions.
            Update position for Krill
            Evaluate each Krill based on its new position
        End for
        Sort all the Krill and find the current best.
End while
Return the best solutions.
End
```

generator in the KH algorithm to initialize the krill positions. Algorithm listing 3 gives the procedure for the Faure sequence improved KH (FaKH) algorithm. The Faure function, faure (*k*, *d*, *b*), takes three parameters; *k* is a non-negative

**Algorithm 3** Pseudocode for the Faure Sequence Improved KH Algorithm

```
Begin
Initialization. Set generation counter; initialize population $NP$ krill; set the foraging speed $V_f$, maximum diffusion speed $D_{max}$, and maximum induced speed $N_{max}$; probability of crossover $pc$.
Evaluate population. Evaluate the krill population based on its position.
For dimension d
    Krill position = (between upper bound and lower bound). * Faure ( )
 End
Krill = evaluate (krill position)
While MaxGeneration do
        Sort all the Krill according to their fitness.
        For all Krill do
            Perform the three motions.
            Update position for Krill
            Evaluate each Krill based on its new position
        End for
        Sort all the Krill and find the current best.
End while
Return the best solutions.
End
```

**Algorithm 4** Pseudocode for the Sobol Sequence Improved KH Algorithm

```
Begin
Initialization. Set generation counter; initialize population $NP$ krill; set the foraging speed $V_f$, maximum diffusion speed $D_{max}$, and maximum induced speed $N_{max}$; probability of crossover $pc$.
Evaluate population. Evaluate the krill population based on its position.
For dimension d
    Krill position = (between upper bound and lower bound). * Sobol ( )
 End
Krill = evaluate (krill position)
While MaxGeneration do
        Sort all the Krill according to their fitness.
        For all Krill do
            Perform the three motions.
            Update position for Krill
            Evaluate each Krill based on its new position
        End for
        Sort all the Krill and find the current best.
End while
Return the best solutions.
End
```

integer for the maximum sequence index, $d$ is a positive integer for the sequence dimension, and $b$ is the sequence base, integer exceeding 1. The output: $s$ is a $d * (k + 1)$ array, with $s(:, i)$ storing $(i + 1)$ Faure sequences.

### 3) SOBOL SEQUENCE-BASED KRILL HERD ALGORITHM

In the same vein, using equation 17, we generate a Sobol sequence that we use to initialize the krill herd position. The Sobol sequence KH algorithm is given in the Algorithm listing 4. The function fnc_getSobolSetMatlab (dim, $N$), provides a Sobol quasi-random distribution, it takes two parameters inputs: dim is the number of variables, the maximum number of variables is 40, and $N$ is the number of samples. The output: $X$ is a $[N \times dim]$ matrix of the quasi-random samples.

## IV. EXPERIMENTAL SETUP

To test the performance of our approach and compare the three proposed quasi-random sequence distribution mechanisms with those from the basic KH algorithm, which used a pseudorandom number generator sequence to initialize krill population, we implemented three versions of the KH algorithm in MATLAB (R2020a), run on Windows 10 OS, Intel Core i7-8550U CPU, 8G RAM. We applied them on a set

**TABLE 3.** Results for functions having a maximum of 10 dimensions.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean |
| F1 | 0 | 9.36E-04 | 9.41E-04 | 2.06E-03 | 9.12E-04 | 1.50E-03 | 2.33E-03 | 6.46E-04 | 6.89E-04 | 1.58E-03 | 9.64E-04 | 7.48E-04 | 1.80E-03 |
| F2 | 0 | 2.22E-02 | 4.10E-02 | 6.23E-02 | 6.70E-02 | 6.08E-02 | 1.60E-01 | 6.85E-02 | 4.06E-02 | 1.24E-01 | 6.28E-02 | 7.67E-02 | 1.50E-01 |
| F3 | 0 | 2.89E-03 | 1.31E-03 | 4.72E-03 | 1.30E-02 | 1.79E-01 | 1.90E-01 | 4.50E-04 | 2.04E-03 | 3.32E-03 | 2.40E-03 | 1.24E-02 | 1.36E-02 |
| F4 | 0 | 9.95E-01 | 3.92E+00 | 5.77E+00 | 2.30E-04 | 3.17E+00 | 9.02E+00 | 1.58E-04 | 3.15E+00 | 8.97E+00 | 9.79E-05 | 1.86E+00 | 1.79E+00 |
| F5 | 0 | 4.33E-01 | 3.44E+02 | 1.65E+02 | 3.64E-03 | 2.50E+00 | 7.10E+00 | 5.49E+00 | 9.27E-01 | 7.72E+00 | 2.36E+00 | 2.65E+01 | 1.55E+01 |
| F6 | 0 | -2.01E+02 | 1.27E+01 | -1.84E+02 | -4.19E+03 | 2.07E+02 | -4.10E+03 | -4.19E+03 | 2.81E+02 | -4.02E+03 | -3.32E+03 | 4.46E+02 | -2.63E+03 |
| F7 | 0 | 2.86E-03 | 1.46E+01 | 9.72E+00 | 5.16E-03 | 1.22E-02 | 1.81E-02 | 8.90E-03 | 2.03E-02 | 2.74E-02 | 3.18E-02 | 5.55E+01 | 8.23E+01 |
| F8 | -0.966015 | -5.75E+00 | 8.66E-01 | -4.46E+00 | -8.42E+00 | 7.75E-01 | -7.69E+00 | -8.54E+00 | 7.90E-01 | -7.02E+00 | -9.08E+00 | 9.22E-01 | -7.24E+00 |
| F9 | 0 | 8.71E-05 | 1.40E-04 | 2.25E-04 | 9.28E-04 | 9.71E-04 | 2.66E-03 | 1.22E-03 | 1.45E-03 | 2.66E-03 | 1.31E-03 | 1.28E-03 | 2.47E-03 |
| F10 | 0 | 2.11E-03 | 1.08E-03 | 3.31E-03 | 5.60E-03 | 3.39E-03 | 1.14E-02 | 6.87E-03 | 4.19E-03 | 1.07E-02 | 6.14E-03 | 3.34E-03 | 9.18E-03 |
| F11 | 0 | 3.36E-11 | 5.02E-08 | 3.21E-08 | 2.57E-10 | 5.66E-08 | 4.99E-08 | 2.40E-10 | 1.75E-08 | 1.70E-08 | 7.55E-10 | 2.18E-08 | 2.71E-08 |
| F12 | -3.86278 | -2.95E+00 | 3.68E-01 | -2.58E+00 | -3.13E+00 | 9.98E-09 | -3.13E+00 | -3.13E+00 | 1.38E-08 | -3.13E+00 | -3.13E+00 | 1.15E-01 | -3.06E+00 |
| F13 | -3.32237 | -2.58E+00 | 2.70E-01 | -2.19E+00 | -3.04E+00 | 1.70E-08 | -3.04E+00 | -3.04E+00 | 1.47E-08 | -3.04E+00 | -3.04E+00 | 3.00E-02 | -3.02E+00 |
| F14 | -3.86278 | -3.77E+00 | 5.21E-01 | -3.34E+00 | -3.86E+00 | 2.44E-01 | -3.79E+00 | -3.86E+00 | 1.05E-04 | -3.86E+00 | -3.86E+00 | 1.16E-04 | -3.86E+00 |
| F15 | -10.5364 | -1.05E+01 | 2.70E+00 | -5.64E+00 | -1.05E+01 | 1.11E-05 | -1.05E+01 | -1.05E+01 | 1.33E-05 | -1.05E+01 | -1.05E+01 | 2.03E+00 | -5.30E+00 |
| F16 | 0 | 4.95E-02 | 3.29E-01 | 3.30E-01 | 4.95E-02 | 6.98E-02 | 1.42E-01 | 7.60E-03 | 9.23E-02 | 9.75E-02 | 3.64E-02 | 3.20E-01 | 3.32E-01 |
| F17 | -1 | -1.00E+00 | 5.16E-01 | -6.00E-01 | -1.00E+00 | 1.89E-04 | -1.00E+00 | -1.00E+00 | 1.84E-04 | -1.00E+00 | -1.00E+00 | 5.16E-01 | -6.00E-01 |
| F18 | 0.397887 | 3.98E-01 | 8.31E-08 | 3.98E-01 | 3.98E-01 | 5.42E-08 | 3.98E-01 | 3.98E-01 | 7.99E-08 | 3.98E-01 | 3.98E-01 | 5.88E-02 | 4.16E-01 |
| F19 | 0 | 4.92E-02 | 2.73E+00 | 3.15E+00 | 2.80E-03 | 2.42E+00 | 1.06E+00 | 1.95E-04 | 1.36E+00 | 5.59E-01 | 2.15E-02 | 2.49E+00 | 1.76E+00 |
| F20 | 3 | 3.00E+00 | 2.19E-07 | 3.00E+00 | 3.00E+00 | 2.56E-07 | 3.00E+00 | 3.00E+00 | 5.86E-08 | 3.00E+00 | 3.00E+00 | 1.37E-07 | 3.00E+00 |

of standard benchmark functions in [44]–[46]. For each of the three variants of the improved KH algorithms, namely, VcKH, SoKH, and FaKH, the corresponding quasi-random number sequence is used to generate the initial krill. Otherwise, the KH algorithm is the same as the basic KH with parameters as presented in [29].

Like most metaheuristic algorithms, the random nature of the KH algorithm makes it difficult to achieve excellent performance in one trial; in reality, multiple trials are needed to judge their performance. In this study, the function evaluations are set to 10,000 for high dimensional functions (F1-F10, F16), and 1000 for low dimensional functions (F11-F20); this is done because the dimension of a function affects the ease of obtaining an optimal solution [45]. The value of $C_t$ is set to 0.5 and $\omega_n, \omega_f$ starts at 0.9. We use two cases to tune the parameters of the algorithms. These two cases are described below,

**Case 1:**

We set the number of replications to 10 runs for the first set of experiments. The krill population is 25, and the maximum number of iterations is set at 200. The dimensions of (F1-F10, F16) are set at 10, 20 and 30, and each algorithm is evaluated accordingly. The effect of these dimensional changes is recorded, as shown in Tables 3, 4, and 5.

**Case 2:**

We set the number of replication to 20 runs, dimension is set to 30, and the maximum number of iterations (MI) and krill population are varied as (50 krill and 300 MI,

**TABLE 4.** Results for functions having a maximum of 20 dimensions.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Best** | **SD** | **Mean** | **Best** | **SD** | **Mean** | **Best** | **SD** | **Mean** | **Best** | **SD** | **Mean** |
| **F1** | **0** | 9.84E-03 | 1.42E+00 | 2.80E+00 | 6.11E-03 | 1.60E-03 | 8.21E-03 | 4.48E-03 | 2.68E-03 | 7.85E-03 | 4.85E-03 | 6.71E-01 | 5.17E-01 |
| **F2** | **0** | 7.40E-05 | 1.64E-02 | 1.69E-02 | 8.27E-02 | 5.35E-02 | 1.61E-01 | 6.27E-02 | 1.44E-01 | 1.70E-01 | 1.17E-01 | 6.49E-02 | 1.91E-01 |
| **F3** | **0** | 3.52E-02 | 6.09E-02 | 9.59E-02 | 1.40E-03 | 1.31E-02 | 1.27E-02 | 6.76E-03 | 1.85E-02 | 2.29E-02 | 2.46E-02 | 3.04E-02 | 5.51E-02 |
| **F4** | **0** | 6.97E+00 | 7.76E+00 | 1.59E+01 | 4.98E+00 | 1.22E-03 | 4.98E+00 | 4.98E+00 | 7.36E-04 | 4.98E+00 | 1.44E-03 | 2.19E+00 | 7.98E-01 |
| **F5** | **0** | 1.42E+01 | 77.23 | 1.01E+02 | 1.81E+01 | 1.20E-01 | 1.82E+01 | 1.68E+01 | 7.60E-01 | 1.84E+01 | 1.72E+01 | 1.73E+01 | 2.51E+01 |
| **F6** | **0** | -5.12E+03 | 1.33E+03 | -3.49E+03 | -8.38E+03 | 5.25E+02 | -8.07E+03 | -8.38E+03 | 5.36E+02 | -8.02E+03 | -5.52E+03 | 1.34E+03 | -3.74E+03 |
| **F7** | **0** | 5.73E+07 | 1.23E+20 | 3.90E+19 | 2.39E+00 | 2.73E+01 | 3.90E+01 | 6.67E-01 | 2.23E+01 | 2.51E+01 | 3.14E+06 | 1.51E+15 | 7.07E+14 |
| **F8** | **-0.966015** | -1.55E+01 | 1.19E+00 | -1.40E+01 | -1.54E+01 | 1.74E+00 | -1.25E+01 | -1.45E+01 | 1.15E+00 | -1.29E+01 | -1.55E+01 | 1.47E+00 | -1.32E+01 |
| **F9** | **0** | 2.97E-02 | 2.43E-01 | 2.27E-01 | 1.99E-02 | 5.33E-02 | 9.09E-02 | 3.64E-02 | 1.25E-01 | 1.23E-01 | 6.26E-02 | 1.89E-01 | 1.60E-01 |
| **F10** | **0** | 2.51E-02 | 6.68E-03 | 3.28E-02 | 2.94E-02 | 5.50E-03 | 3.57E-02 | 2.55E-02 | 6.62E-03 | 3.72E-02 | 2.60E-02 | 6.63E-03 | 3.87E-02 |
| **F16** | **0** | 2.12E+01 | 3.76E+01 | 7.74E+01 | 1.12E+01 | 1.36E+01 | 3.06E+01 | 1.18E-01 | 2.72E+01 | 3.07E+01 | 2.90E+01 | 2.40E+01 | 6.58E+01 |

**TABLE 5.** Results of functions having a maximum 30 dimensions.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Best** | **SD** | **Mean** | **Best** | **SD** | **Mean** | **Best** | **SD** | **Mean** | **Best** | **SD** | **Mean** |
| **F1** | **0** | 2.82E+00 | 1.55E+00 | 5.81E+00 | 1.15E-02 | 2.48E-03 | 1.57E-02 | 9.79E-03 | 4.19E-01 | 1.50E-01 | 1.81E-02 | 8.20E-01 | 8.90E-01 |
| **F2** | **0** | 1.51E-01 | 1.22E-01 | 2.76E-01 | 1.41E-01 | 7.15E-02 | 2.26E-01 | 1.45E-01 | 7.25E-02 | 2.31E-01 | 1.54E-01 | 8.70E-02 | 2.61E-01 |
| **F3** | **0** | 1.27E-01 | 9.75E-02 | 2.56E-01 | 1.53E-03 | 9.44E-03 | 1.36E-02 | 1.82E-03 | 1.44E-02 | 1.73E-02 | 2.34E-02 | 2.26E-02 | 5.14E-02 |
| **F4** | **0** | 1.15E+01 | 7.32E+00 | 1.90E+01 | 7.47E+00 | 2.80E-03 | 7.47E+00 | 7.47E+00 | 3.80E-03 | 7.47E+00 | 2.97E-03 | 4.03E-03 | 9.32E-03 |
| **F5** | **0** | 2.96E+01 | 3.02E+02 | 2.04E+02 | 2.81E+01 | 1.74E-01 | 2.83E+01 | 2.68E+01 | 7.92E-01 | 2.87E+01 | 2.54E+01 | 3.81E+01 | 6.01E+01 |
| **F6** | **0** | -9.00E+03 | 2.02E+03 | -5.47E+03 | -1.26E+04 | 8.62E+02 | -1.18E+04 | -1.26E+04 | 7.35E+02 | -1.19E+04 | -9.74E+03 | 1.82E+03 | -7.18E+03 |
| **F7** | **0** | 2.64E+22 | 6.53E+34 | 3.00E+34 | 7.57E+01 | 1.79E+05 | 1.10E+05 | 7.60E+01 | 8.26E+08 | 4.01E+08 | 1.49E+24 | 1.27E+33 | 6.64E+32 |
| **F8** | **-0.966015** | -2.12E+01 | 2.15E+00 | -1.84E+01 | -2.04E+01 | 1.91E+00 | -1.73E+01 | -2.06E+01 | 2.47E+00 | -1.72E+01 | -2.13E+01 | 1.88E+00 | -1.91E+01 |
| **F9** | **0** | 1.15E+00 | 2.93E+00 | 3.37E+00 | 1.04E-01 | 8.48E-01 | 1.02E+00 | 1.90E-01 | 1.61E+00 | 1.36E+00 | 1.72E-01 | 5.45E-01 | 1.17E+00 |
| **F10** | **0** | 3.52E-02 | 2.26E-02 | 8.21E-02 | 4.80E-02 | 1.74E-02 | 7.43E-02 | 5.31E-02 | 2.03E-02 | 8.13E-02 | 5.57E-02 | 1.84E-02 | 7.95E-02 |
| **F16** | **0** | 1.66E+02 | 6.01E+01 | 2.40E+02 | 2.61E+01 | 8.31E+01 | 1.32E+02 | 3.58E+01 | 5.89E+01 | 1.52E+02 | 1.45E+02 | 8.81E+01 | 2.72E+02 |

100 krill and 400 MI, 150 krill, and 500 MI, and 200 krill and 600 MI, respectively). The effect of the number of krill and a maximum number of iterations on the performance of the four (4) algorithms are presented in Tables 6, 7, 8, and 9. In all the experiments, the value of the best run, the mean value of the objective function for all the runs, CPU time, and standard deviation are recorded for each algorithm after 20 experimental trials.

### A. BENCHMARKED TEST FUNCTIONS

We chose 20 commonly used and standard mathematical benchmark optimization functions available in the literature to test the performance of the proposed algorithms. The functions are given in Table 2; they are a combination of multimodal, unimodal, non-separable, and separable functions. F1-F10, F16 are high dimensional functions (which we varied from 10 to 30) while

**TABLE 6.** Results for 50 krill and maximum iteration 300.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean |
| F1 | 0 | 3.29E-03 | 9.11E-01 | 1.70E+00 | 2.49E-03 | 1.28E+00 | 2.13E+00 | 4.15E-03 | 1.31E+00 | 1.59E+00 | 2.38E-03 | 1.42E-02 | 7.06E-03 |
| F2 | 0 | 4.51E-02 | 2.70E-02 | 8.21E-02 | 2.52E-02 | 2.84E-02 | 6.00E-02 | 4.00E-02 | 2.14E-02 | 5.89E-02 | 4.91E-02 | 2.50E-02 | 8.52E-02 |
| F3 | 0 | 3.59E-02 | 3.73E-02 | 8.49E-02 | 2.10E-03 | 9.72E-03 | 9.14E-03 | 2.02E-03 | 1.17E-02 | 1.46E-02 | 3.73E-04 | 1.18E-02 | 1.79E-02 |
| F4 | 0 | 5.97E+00 | 6.62E+00 | 1.24E+01 | 3.46E-04 | 2.66E-04 | 7.39E-04 | 3.16E-06 | 1.06E-05 | 2.16E-05 | 3.52E-04 | 3.49E-04 | 9.76E-04 |
| F5 | 0 | 2.12E+01 | 8.40E+01 | 7.49E+01 | 5.30E-05 | 6.60E-05 | 1.53E-04 | 6.07E+00 | 4.06E+00 | 1.19E+01 | 2.61E+01 | 1.23E+01 | 3.07E+01 |
| F6 | 0 | -7.74E+03 | 1.69E+03 | -5.48E+03 | -1.26E+04 | 2.68E-03 | -1.26E+04 | -1.26E+04 | 6.24E+00 | -1.26E+04 | -7.30E+03 | 1.67E+03 | -5.21E+03 |
| F7 | 0 | 4.55E+18 | 6.23E+33 | 2.04E+33 | 9.18E-03 | 1.23E+05 | 2.78E+04 | 9.01E-03 | 4.40E+08 | 9.88E+07 | 4.73E+21 | 2.71E+36 | 8.12E+35 |
| F8 | -0.966015 | -2.41E+01 | 1.90E+00 | -2.12E+01 | -2.41E+01 | 2.47E+00 | -2.02E+01 | -2.36E+01 | 1.98E+00 | -2.08E+01 | -2.40E+01 | 2.38E+00 | -2.07E+01 |
| F9 | 0 | 4.75E-02 | 1.00E-01 | 1.42E-01 | 9.40E-03 | 1.27E-02 | 2.90E-02 | 1.51E-02 | 1.23E-02 | 3.15E-02 | 4.81E-02 | 9.40E-02 | 1.26E-01 |
| F10 | 0 | 1.74E-02 | 4.80E-03 | 2.55E-02 | 1.56E-02 | 4.30E-03 | 2.34E-02 | 1.31E-02 | 5.55E-03 | 2.48E-02 | 1.63E-02 | 6.41E-03 | 2.66E-02 |
| F16 | 0 | 6.16E+01 | 4.79E+01 | 1.40E+02 | 1.97E+01 | 5.65E+01 | 9.86E+01 | 1.99E+01 | 4.09E+01 | 7.00E+01 | 6.19E+01 | 5.83E+01 | 1.52E+02 |

**TABLE 7.** Results for 100 krill and maximum iteration 400.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean |
| F1 | 0 | 7.50E-04 | 7.25E-01 | 4.46E-01 | 8.18E-04 | 3.02E-04 | 1.15E-03 | 5.56E-04 | 3.48E-04 | 1.07E-03 | 5.19E-04 | 2.70E-04 | 9.45E-04 |
| F2 | 0 | 1.78E-02 | 1.81E-02 | 3.73E-02 | 1.75E-02 | 1.04E-02 | 2.96E-02 | 2.04E-02 | 1.83E-02 | 3.51E-02 | 1.72E-02 | 2.24E-02 | 3.94E-02 |
| F3 | 0 | 1.04E-02 | 1.73E-02 | 3.71E-02 | 8.37E-04 | 6.98E-03 | 8.12E-03 | 2.34E-05 | 9.22E-03 | 7.84E-03 | 3.05E-03 | 1.02E-02 | 1.45E-02 |
| F4 | 0 | 1.70E-04 | 6.87E+00 | 7.76E+00 | 9.50E-05 | 6.02E-05 | 1.66E-04 | 2.99E-06 | 3.21E-06 | 5.28E-06 | 5.19E-05 | 5.27E-05 | 1.66E-04 |
| F5 | 0 | 2.48E+01 | 5.85E+01 | 4.41E+01 | 3.07E-05 | 1.13E-04 | 1.66E-04 | 5.90E-01 | 3.20E-01 | 1.02E+00 | 2.50E+01 | 1.78E+01 | 3.18E+01 |
| F6 | 0 | -9.19E+03 | 1.44E+03 | -6.44E+03 | -1.26E+04 | 3.69E-04 | -1.26E+04 | -1.26E+04 | 1.15E-01 | -1.26E+04 | -9.21E+03 | 1.46E+03 | -7.00E+03 |
| F7 | 0 | 3.09E+18 | 9.90E+38 | 2.23E+38 | 2.65E+01 | 4.15E+04 | 3.67E+04 | 1.14E+01 | 3.95E+04 | 2.89E+04 | 3.13E+24 | 1.21E+37 | 5.63E+36 |
| F8 | -0.966015 | -2.56E+01 | 1.67E+00 | -2.26E+01 | -2.63E+01 | 2.27E+00 | -2.21E+01 | -2.47E+01 | 1.82E+00 | -2.09E+01 | -2.58E+01 | 1.72E+00 | -2.29E+01 |
| F9 | 0 | 4.60E-03 | 6.95E-03 | 1.34E-02 | 5.12E-03 | 3.69E-03 | 1.26E-02 | 5.02E-03 | 3.12E-03 | 9.77E-03 | 4.76E-03 | 6.70E-03 | 1.48E-02 |
| F10 | 0 | 4.09E-03 | 1.84E-03 | 7.02E-03 | 5.77E-03 | 2.72E-03 | 1.15E-02 | 6.88E-03 | 2.70E-03 | 1.14E-02 | 6.79E-03 | 2.76E-03 | 1.04E-02 |
| F16 | 0 | 2.63E+01 | 5.43E+01 | 9.04E+01 | 1.84E+01 | 3.03E+01 | 5.73E+01 | 1.38E+01 | 4.01E+01 | 5.63E+01 | 4.57E+01 | 3.03E+01 | 9.05E+01 |

F11-F16, F17-F20 are low dimensional functions (less than 10).

### B. RESULTS AND DISCUSSION

In this section, we present and discuss the results obtained in our experiments. As our experimental setup is divided into two cases, we present our results accordingly.

#### 1) CASE 1: EXPERIMENTS WITH LOW DIMENSIONAL TEST FUNCTIONS

The numerical results of the experiments conducted for this study to evaluate the algorithms' performance are presented in Tables 3, 4, and 5. For each test function (F1-F20), all four (4) algorithms are evaluated, and the results for the best run, mean value of the objective function for all the

**TABLE 8.** Results for 150 krill and maximum iteration 500.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean |
| F1 | 0 | 2.28E-04 | 4.62E-01 | 1.51E-01 | 2.25E-04 | 2.11E-04 | 4.65E-04 | 2.33E-04 | 1.97E-04 | 4.20E-04 | 1.71E-04 | 1.56E-04 | 3.84E-04 |
| F2 | 0 | 1.32E-02 | 1.10E-02 | 2.64E-02 | 1.25E-02 | 4.98E-03 | 1.92E-02 | 1.11E-02 | 4.14E-03 | 1.81E-02 | 1.21E-02 | 1.53E-02 | 3.09E-02 |
| F3 | 0 | 1.05E-02 | 8.87E-03 | 2.09E-02 | 2.80E-04 | 4.59E-03 | 5.02E-03 | 4.23E-04 | 5.81E-03 | 6.63E-03 | 4.03E-03 | 6.47E-03 | 1.21E-02 |
| F4 | 0 | 9.95E-01 | 2.23E+00 | 4.88E+00 | 2.93E-05 | 1.90E-05 | 5.49E-05 | 2.36E-07 | 7.71E-07 | 1.76E-06 | 2.94E-05 | 1.58E-05 | 5.92E-05 |
| F5 | 0 | 2.69E+01 | 1.66E+01 | 3.35E+01 | 3.02E-05 | 3.89E-04 | 3.59E-04 | 6.29E-02 | 3.39E-02 | 1.13E-01 | 2.52E+01 | 9.17E-01 | 2.74E+01 |
| F6 | 0 | -8.64E+03 | 9.58E+02 | -6.86E+03 | -1.26E+04 | 6.29E-04 | -1.26E+04 | -1.26E+04 | 2.48E-04 | -1.26E+04 | -1.04E+04 | 1.45E+03 | -7.15E+03 |
| F7 | 0 | 5.02E+20 | 5.48E+37 | 1.39E+37 | 2.60E-02 | 1.14E+01 | 1.29E+01 | 3.15E-02 | 9.06E+00 | 1.30E+01 | 1.29E+21 | 9.45E+36 | 4.55E+36 |
| F8 | -0.966015 | -2.63E+01 | 2.43E+00 | -2.30E+01 | -2.54E+01 | 1.59E+00 | -2.19E+01 | -2.75E+01 | 2.00E+00 | -2.24E+01 | -2.74E+01 | 2.28E+00 | -2.35E+01 |
| F9 | 0 | 2.37E-03 | 1.87E-03 | 4.98E-03 | 2.16E-03 | 1.60E-03 | 4.14E-03 | 1.72E-03 | 1.23E-03 | 3.21E-03 | 1.08E-03 | 3.06E-03 | 5.12E-03 |
| F10 | 0 | 3.52E-03 | 1.36E-03 | 5.44E-03 | 4.58E-03 | 1.29E-03 | 6.64E-03 | 5.00E-03 | 1.61E-03 | 7.91E-03 | 3.01E-03 | 1.98E-03 | 7.32E-03 |
| F16 | 0 | 3.28E+01 | 2.29E+01 | 6.06E+01 | 2.90E+00 | 9.71E+00 | 1.16E+01 | 2.83E+00 | 4.45E+00 | 9.64E+00 | 1.57E+01 | 2.54E+01 | 5.05E+01 |

**TABLE 9.** Results for 200 krill and maximum iteration 600.

| Fn | Global | KH | | | VcKH | | | FaKH | | | SoKH | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean | Best | SD | Mean |
| F1 | 0 | 1.18E-04 | 2.58E-01 | 5.84E-02 | 1.05E-04 | 4.65E-05 | 1.88E-04 | 8.83E-05 | 5.06E-05 | 1.85E-04 | 9.60E-05 | 6.23E-05 | 1.91E-04 |
| F2 | 0 | 8.43E-03 | 1.91E-02 | 2.50E-02 | 6.75E-03 | 5.24E-03 | 1.51E-02 | 6.87E-03 | 9.12E-03 | 1.52E-02 | 9.73E-03 | 1.51E-02 | 2.42E-02 |
| F3 | 0 | 4.72E-03 | 6.95E-03 | 1.50E-02 | 1.59E-04 | 3.04E-03 | 4.11E-03 | 7.21E-05 | 4.61E-03 | 4.54E-03 | 4.39E-04 | 6.73E-03 | 7.52E-03 |
| F4 | 0 | 2.27E-05 | 2.97E+00 | 2.39E+00 | 1.48E-05 | 9.54E-06 | 2.73E-05 | 1.69E-05 | 6.06E-06 | 2.58E-05 | 1.64E-05 | 8.85E-06 | 2.74E-05 |
| F5 | 0 | 2.37E+01 | 1.84E+01 | 3.28E+01 | 3.44E-06 | 3.40E-04 | 2.91E-04 | 3.56E-02 | 1.66E-02 | 6.76E-02 | 2.66E+01 | 5.63E-01 | 2.74E+01 |
| F6 | 0 | -8.54E+03 | 9.86E+02 | -6.92E+03 | -1.26E+04 | 7.38E-05 | -1.26E+04 | -1.26E+04 | 7.95E-05 | -1.26E+04 | -1.05E+04 | 1.19E+03 | -7.90E+03 |
| F7 | 0 | 5.34E+23 | 1.81E+37 | 5.64E+36 | 3.51E-03 | 2.67E+00 | 9.65E-01 | 2.17E-03 | 2.74E+00 | 1.53E+00 | 3.08E+23 | 5.43E+36 | 2.55E+36 |
| F8 | -0.966015 | -2.58E+01 | 1.89E+00 | -2.32E+01 | -2.59E+01 | 2.31E+00 | -2.24E+01 | -2.51E+01 | 1.53E+00 | -2.26E+01 | -2.65E+01 | 2.24E+00 | -2.31E+01 |
| F9 | 0 | 6.25E-04 | 9.28E-04 | 2.01E-03 | 8.94E-04 | 6.27E-04 | 1.63E-03 | 1.13E-03 | 7.36E-04 | 2.20E-03 | 9.16E-04 | 5.78E-04 | 2.06E-03 |
| F10 | 0 | 3.03E-03 | 1.04E-03 | 4.48E-03 | 2.48E-03 | 1.40E-03 | 5.08E-03 | 3.51E-03 | 1.00E-03 | 5.23E-03 | 2.96E-03 | 1.61E-03 | 4.95E-03 |
| F16 | 0 | 1.96E+01 | 1.83E+01 | 4.04E+01 | 5.92E-01 | 3.70E+00 | 6.37E+00 | 1.83E+00 | 2.08E+00 | 5.05E+00 | 1.07E+01 | 2.09E+01 | 4.32E+01 |

runs, CPU time, and standard deviations are recorded. In the literature, it was stated that the performance of most metaheuristic algorithms would diminish or scale down as the test function's dimension increases [47]; to verify this influence, the dimension of function (F1-F10, F16) is varied at 10, 20, and 30, respectively.

The numerical results for the test benchmark function dimension that is less or equal to 10 are given in Table 3. We see clearly from the table that the standard deviations (SD) for VcKH, FaKH, and SoKH are less than that of the basic KH algorithm. These low SD show that the different values of the objective function obtained at different runs of

**TABLE 10.** CPU time for Case 1.

| Fn | Dimension | KH Time (Sec) | VcKH Time (Sec) | FaKH Time (Sec) | SoKH Time (Sec) |
|---|---|---|---|---|---|
| F1 | 10 | 4.764 | 8.486 | 9.582 | 9.537 |
| | 20 | 9.791 | 4.206 | 4.234 | 9.587 |
| | 30 | 9.915 | 4.312 | 4.246 | 9.669 |
| F2 | 10 | 3.701 | 9.207 | 9.287 | 9.819 |
| | 20 | 9.243 | 4.049 | 4.189 | 9.437 |
| | 30 | 9.319 | 4.150 | 4.462 | 9.777 |
| F3 | 10 | 4.110 | 8.942 | 9.509 | 9.365 |
| | 20 | 9.356 | 4.326 | 4.098 | 9.114 |
| | 30 | 9.144 | 4.120 | 4.554 | 10.121 |
| F4 | 10 | 4.028 | 9.192 | 9.454 | 9.357 |
| | 20 | 9.290 | 4.187 | 9.428 | 9.684 |
| | 30 | 9.144 | 4.120 | 4.554 | 10.121 |
| F5 | 10 | 4.102 | 8.997 | 9.093 | 9.334 |
| | 20 | 8.687 | 4.428 | 9.645 | 9.651 |
| | 30 | 8.639 | 4.057 | 9.595 | 9.482 |
| F6 | 10 | 3.931 | 10.304 | 9.457 | 9.824 |
| | 20 | 8.464 | 4.296 | 9.349 | 9.525 |
| | 30 | 8.773 | 4.307 | 9.520 | 10.495 |
| F7 | 10 | 4.214 | 9.168 | 9.214 | 9.822 |
| | 20 | 7.807 | 4.096 | 9.572 | 9.510 |
| | 30 | 8.298 | 4.011 | 9.259 | 10.192 |
| F8 | 10 | 4.192 | 9.859 | 9.792 | 10.160 |
| | 20 | 9.389 | 4.771 | 11.256 | 10.672 |
| | 30 | 9.334 | 4.475 | 10.397 | 10.700 |
| F9 | 10 | 4.305 | 9.714 | 9.601 | 10.101 |
| | 20 | 10.280 | 4.886 | 10.666 | 12.016 |
| | 30 | 11.962 | 5.664 | 13.163 | 15.316 |
| F10 | 10 | 3.960 | 8.564 | 8.575 | 9.164 |
| | 20 | 8.534 | 4.146 | 9.635 | 10.465 |
| | 30 | 8.703 | 4.105 | 9.829 | 10.101 |
| F11 | 10 | 7.884 | 8.464 | 9.052 | 9.362 |
| F12 | 10 | 8.671 | 9.327 | 9.499 | 9.850 |
| F13 | 10 | 9.623 | 9.771 | 9.994 | 10.243 |
| F14 | 10 | 8.504 | 9.542 | 9.956 | 10.226 |
| F15 | 10 | 9.037 | 9.732 | 10.129 | 10.436 |
| F16 | 10 | 8.221 | 8.956 | 9.531 | 9.635 |
| Fn | Dimension | KH Time (Sec) | VcKH Time (Sec) | FaKH Time (Sec) | SoKH Time (Sec) |
| | 20 | 8.956 | 4.319 | 9.742 | 10.992 |
| | 30 | 4.167 | 4.195 | 9.602 | 10.438 |
| F17 | 10 | 8.815 | 9.049 | 9.256 | 9.171 |
| F18 | 10 | 9.069 | 8.194 | 9.097 | 8.724 |
| F19 | 10 | 8.635 | 8.993 | 8.538 | 9.240 |
| F20 | 10 | 8.589 | 8.982 | 9.004 | 9.373 |

**TABLE 11.** CPU time for Case 2.

| Fn | Number of Krill and MI | KH Time (Sec) | VcKH Tim e(Sec) | FaKH Time (Sec) | SoKH Time (Sec) |
|---|---|---|---|---|---|
| F1 | 50 and 300 | 49.027 | 48.667 | 48.279 | 46.541 |
| | 100 and 400 | 171.297 | 170.350 | 182.355 | 263.739 |
| | 150 and 500 | 430.018 | 418.395 | 427.571 | 418.447 |
| | 200 and 600 | 880.041 | 852.766 | 883.762 | 855.202 |
| F2 | 50 and 300 | 47.690 | 45.663 | 48.045 | 45.757 |
| | 100 and 400 | 168.061 | 237.668 | 174.467 | 168.026 |
| | 150 and 500 | 427.971 | 416.484 | 434.209 | 223.957 |
| | 200 and 600 | 878.578 | 868.056 | 868.167 | 423.870 |
| F3 | 50 and 300 | 49.136 | 49.193 | 47.859 | 20.696 |
| | 100 and 400 | 175.026 | 170.844 | 175.742 | 83.778 |
| | 150 and 500 | 432.088 | 428.239 | 428.160 | 207.236 |
| | 200 and 600 | 889.281 | 892.824 | 1261.400 | 443.148 |
| F4 | 50 and 300 | 46.099 | 44.417 | 41.873 | 21.805 |
| | 100 and 400 | 172.939 | 172.651 | 157.181 | 83.764 |
| | 150 and 500 | 427.514 | 421.177 | 385.787 | 206.954 |
| | 200 and 600 | 456.404 | 853.811 | 792.667 | 434.283 |
| F5 | 50 and 300 | 23.852 | 41.681 | 47.858 | 23.007 |
| | 100 and 400 | 92.056 | 153.585 | 177.633 | 86.273 |
| | 150 and 500 | 226.862 | 387.045 | 431.392 | 208.848 |
| | 200 and 600 | 437.231 | 799.622 | 867.734 | 433.572 |
| F6 | 50 and 300 | 22.365 | 46.267 | 23.525 | 21.992 |
| | 100 and 400 | 84.819 | 174.048 | 83.085 | 83.471 |
| | 150 and 500 | 215.962 | 431.166 | 216.263 | 208.431 |
| | 200 and 600 | 438.888 | 842.323 | 423.874 | 426.219 |
| F7 | 50 and 300 | 21.804 | 44.269 | 21.506 | 21.846 |
| | 100 and 400 | 80.887 | 167.370 | 84.405 | 80.776 |
| | 150 and 500 | 219.549 | 356.533 | 208.642 | 208.418 |
| | 200 and 600 | 433.498 | 424.976 | 430.669 | 422.798 |
| F8 | 50 and 300 | 24.161 | 24.323 | 24.001 | 23.486 |
| | 100 and 400 | 90.647 | 89.162 | 191.395 | 176.842 |
| | 150 and 500 | 227.078 | 215.351 | 617.638 | 439.864 |
| | 200 and 600 | 455.289 | 442.296 | 879.988 | 876.678 |
| F9 | 50 and 300 | 32.088 | 33.172 | 65.380 | 67.068 |
| | 100 and 400 | 112.814 | 108.054 | 223.668 | 222.998 |
| | 150 and 500 | 258.022 | 254.317 | 509.335 | 630.965 |
| | 200 and 600 | 517.747 | 495.558 | 999.064 | 1002.487 |
| F10 | 50 and 300 | 22.582 | 21.992 | 45.129 | 68.216 |
| | 100 and 400 | 89.657 | 167.573 | 169.894 | 168.045 |
| | 150 and 500 | 448.294 | 412.126 | 456.748 | 415.081 |
| | 200 and 600 | 428.879 | 846.644 | 851.008 | 838.310 |
| F16 | 50 and 300 | 24.126 | 45.759 | 48.082 | 47.020 |
| | 100 and 400 | 89.657 | 167.573 | 169.894 | 168.045 |
| | 150 and 500 | 227.377 | 426.552 | 428.501 | 425.503 |
| | 200 and 600 | 449.214 | 865.306 | 850.743 | 863.392 |

the algorithms are close to the best run value; hence, this means better results are obtained for each successive run. This is because, with each subsequent run, the krill become more uniformly distributed across the search space, which is in line with the characteristics of low-discrepancy sequences [39] used to initialize the krill population. It is also clear that the values of the best runs for the three improved quasi-random sequence-based KH algorithms are less than that of the basic KH algorithm with few exceptions (F2, F7, F9). These best run values show that the algorithm's performance was significantly improved by the initial uniform distribution of the krill across the search space using the low-discrepancy sequences.

From Table 4, it is noticeable that the performance of the algorithms decreases as the dimension of the problem is increased. From Table 4, it can also be seen that the value of the best run is significantly far from the global optimum of the respective functions (F4, F7, F16). However, the performance of the three improved KH algorithms is better than the that of the basic KH in most cases here, and the standard deviations are significantly lower than for the basic KH algorithm in most of the instances presented.

Overall, the FaKH algorithm performed better in seven out of eleven functions. Comparing Tables 3 and 4, we see that the values for the best krill are significantly lower (closer to optimal) in Table 3 than in Table 4. This can be attributed to the performance of the algorithms decreasing as the problem dimension increases.

In Table 5, for all the algorithms, the value of the best run is significantly close to the global optimal for F1-F3, F9. However, only SoKH did well in F4; by contrast, the value of the best run for all other algorithms is significantly far

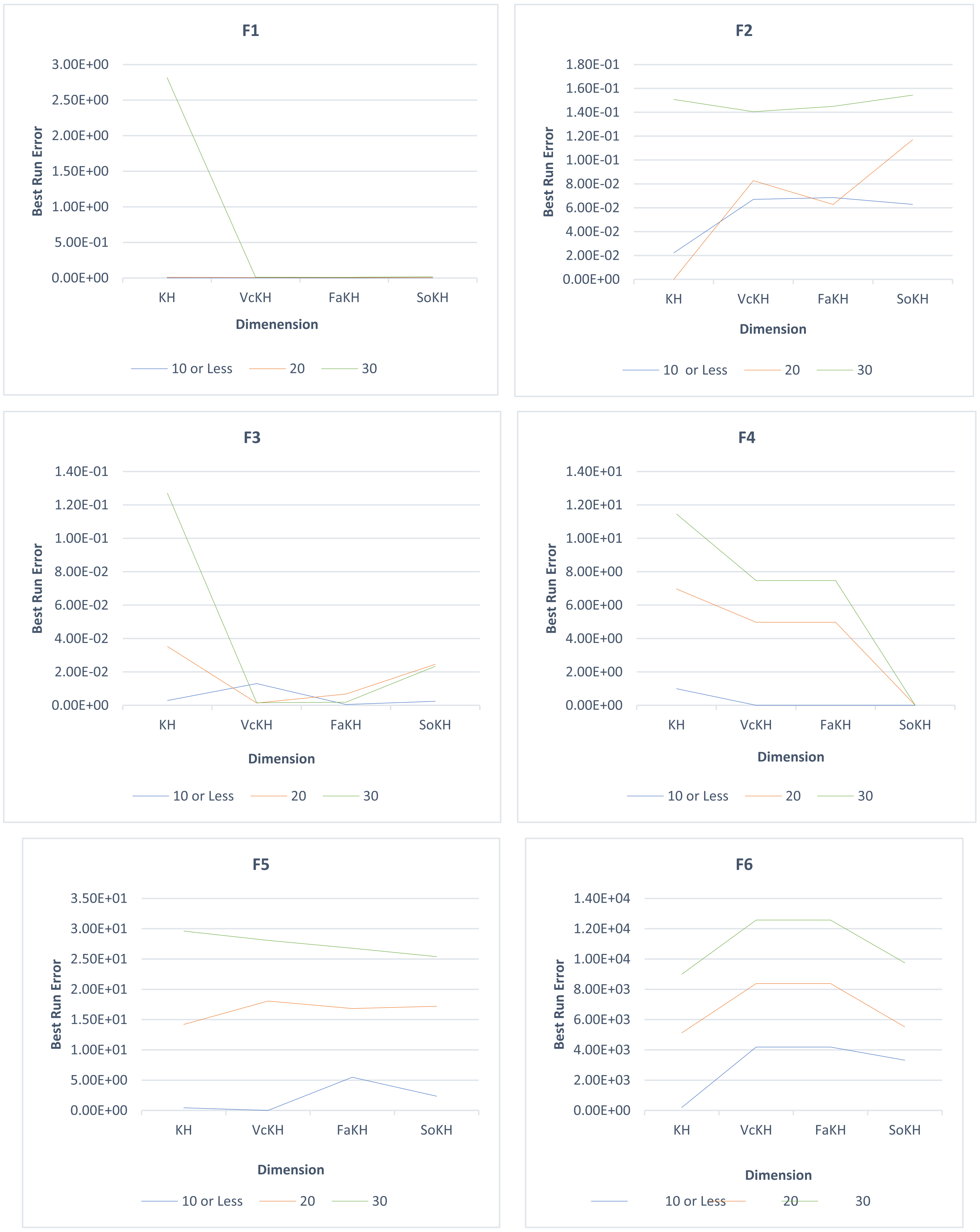
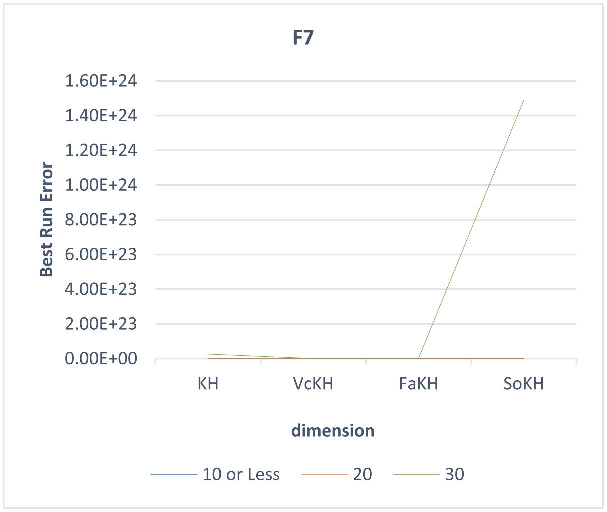
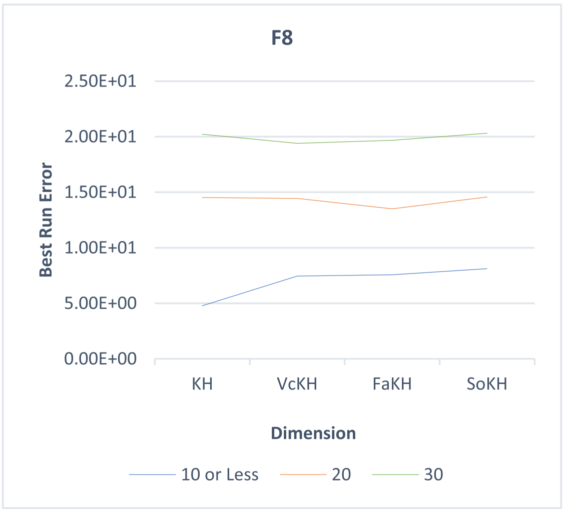
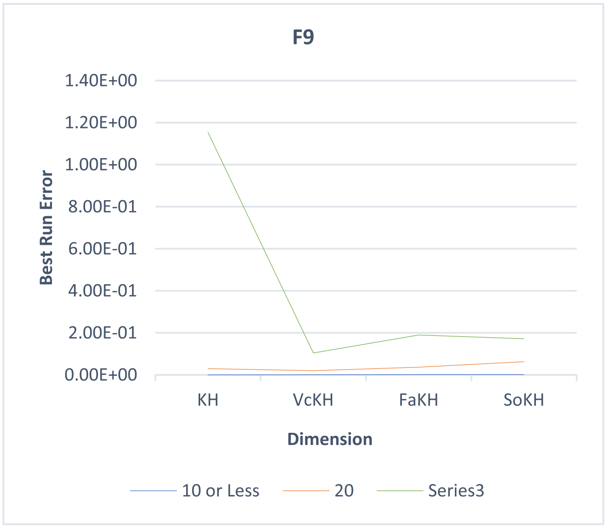
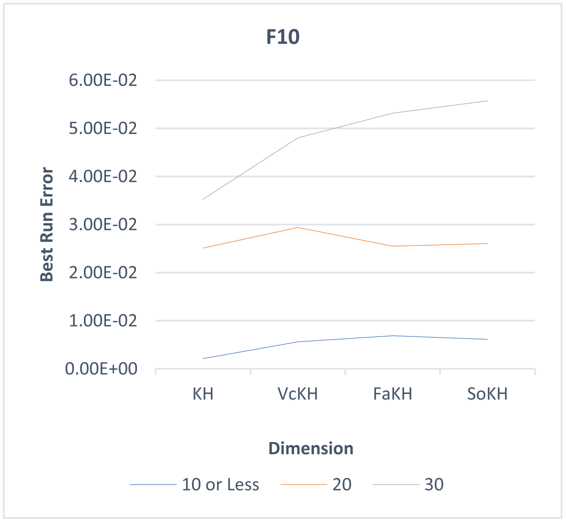


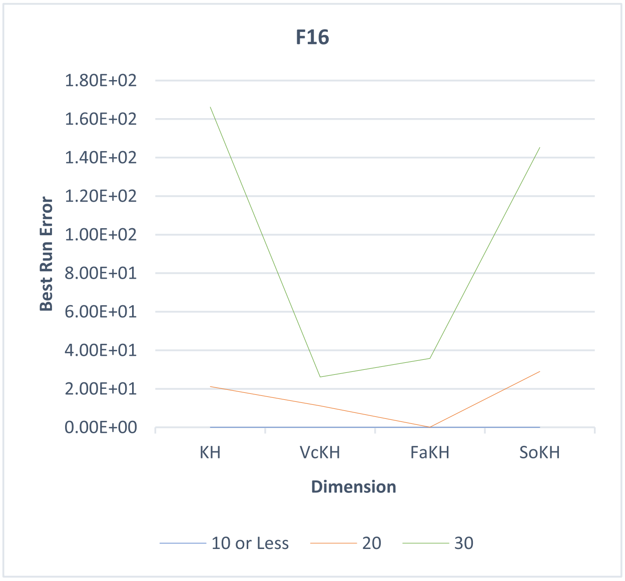
FIGURE 7. Best-run error curves for 25 krill and maximum iteration 200.











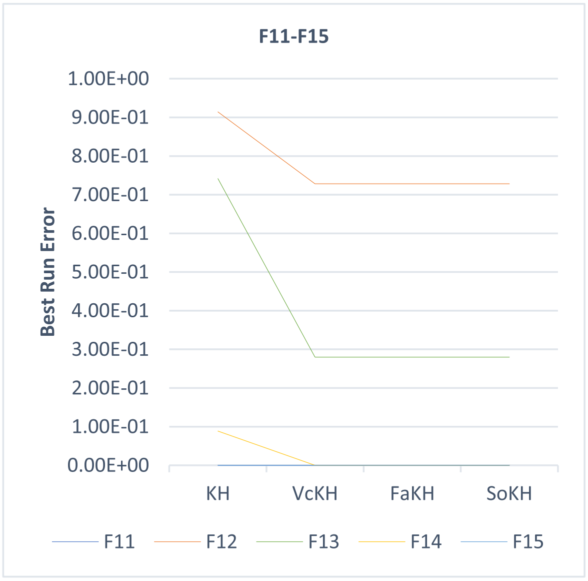


**FIGURE 7.** *(Continued.)* **Best-run error curves for 25 krill and maximum iteration 200.**

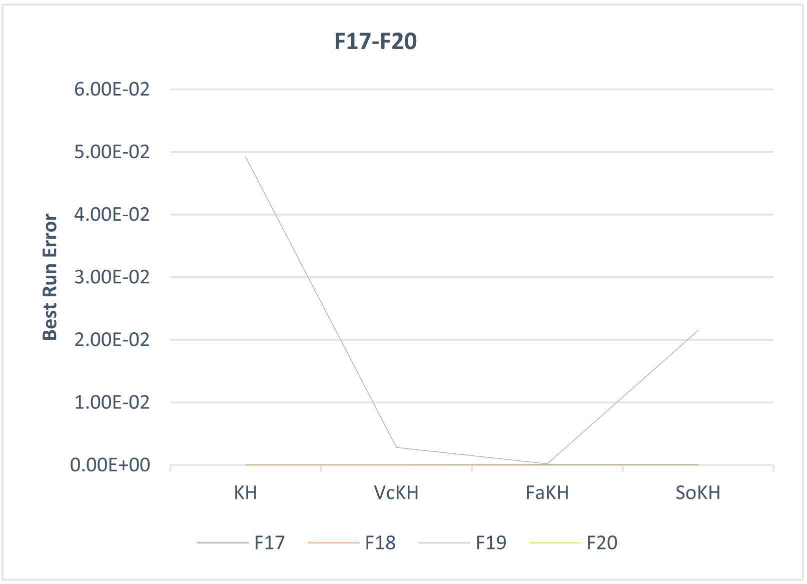


**FIGURE 7.** ***(Continued.)*** **Best-run error curves for 25 krill and maximum iteration 200.**

from the global optimal for F5-F8, F16. Overall, SoKH outperforms all the other algorithms (doing well in 6 out of 11 functions). This shows that, as the dimension increased, the performance of the algorithms reduced. Also, comparing Tables 3, 4, and 5, shows that the best-run values are higher (far from global optimal) in Table 5 than in Tables 3 and 4. Nevertheless, in cases where our improved algorithm did well, it outperforms the basic KH algorithm.

### 2) CASE 2: EXPERIMENTS WITH HIGH DIMENSIONAL TEST FUNCTIONS

We have seen in case 1 the effect of problem dimension on the performance of the algorithms. Next, we evaluated the impact of the size of the krill population and maximum iteration (MI) on the performance of the algorithms. Looking at Table 6, we see significant improvements in all the algorithms as compared to the case in Table 5. The dimensions being the same for both tables gives a basis for comparison and clearly shows the positive effect of increasing the krill herd to 50 and iterations to 300. A significant improvement can be seen in the VcKH algorithm, which did well in 8 out of the 11 functions to outperform the remaining three algorithms.

Increasing the krill population size to 100 and maximum iteration to 400, Table 7 shows that the algorithms performed well for all functions except for F6-F8, and F16. Only the VcKH and FaKH algorithms performed well in F5 and here outperformed the rest of the algorithms. While Table 7 showed improvement in the performance of the algorithm, comparing it with Table 6 shows that the performance of VcKH reduced from 8 out of 11 to 7 out of 11 with increased population size and iterations. The performance of the other algorithms remained unchanged. The best run value of the algorithms for each function showed significant improvement, being close to the global optimal.

In Table 8, it can be seen that VcKH and FaKH outperformed the remaining algorithms in 8 out of 11 functions. The best-run values of the algorithm for each function showed significant improvement as compared to results in Table 7; this showed the positive effect of increasing krill herd to 150 and MI to 500. Table 9 shows a similar pattern to previous observations. The performance of SoKH algorithm has been consistent throughout the second set of experiments, doing well or poorly for the same set of functions.

The CPU time for all experiments run in case 1 is given in Table 10. We see clearly that the KH has less CPU time as compared with the other three algorithms in most cases when the dimension is low. This can be attributed to the extra cost of implementing the function calls that generate the Van der Corput, Faure, and Sobol sequences. Unlike in the case of the random number generator, which is an in-built function in MATLAB, and so it has less effect on the CPU time.

SoKH consumed more CPU time as compared to FaKH and VcKH because the Sobol function is known to generate sequences in more computational time than do the Faure or Van der Corput [49]. As the dimension gets bigger, we see the FaKH algorithm and VcKH having less CPU time. No clear pattern can be deduced from Table 11, which shows the CPU time for case 2. Here, all algorithms consumed more CPU time, which increases as the number of krill and MI increases.

In addition, to further evaluate the performance accuracy of the proposed improved KH algorithms, we used the best-run error metrics. The best run error is defined as the absolute difference between the best-run value $f(x^*)$ obtained for each algorithm and the actual global minimum $f(x)$ for each function. Mathematically, the best error function is defined as $error = |f(x^*) - f(x)|$.

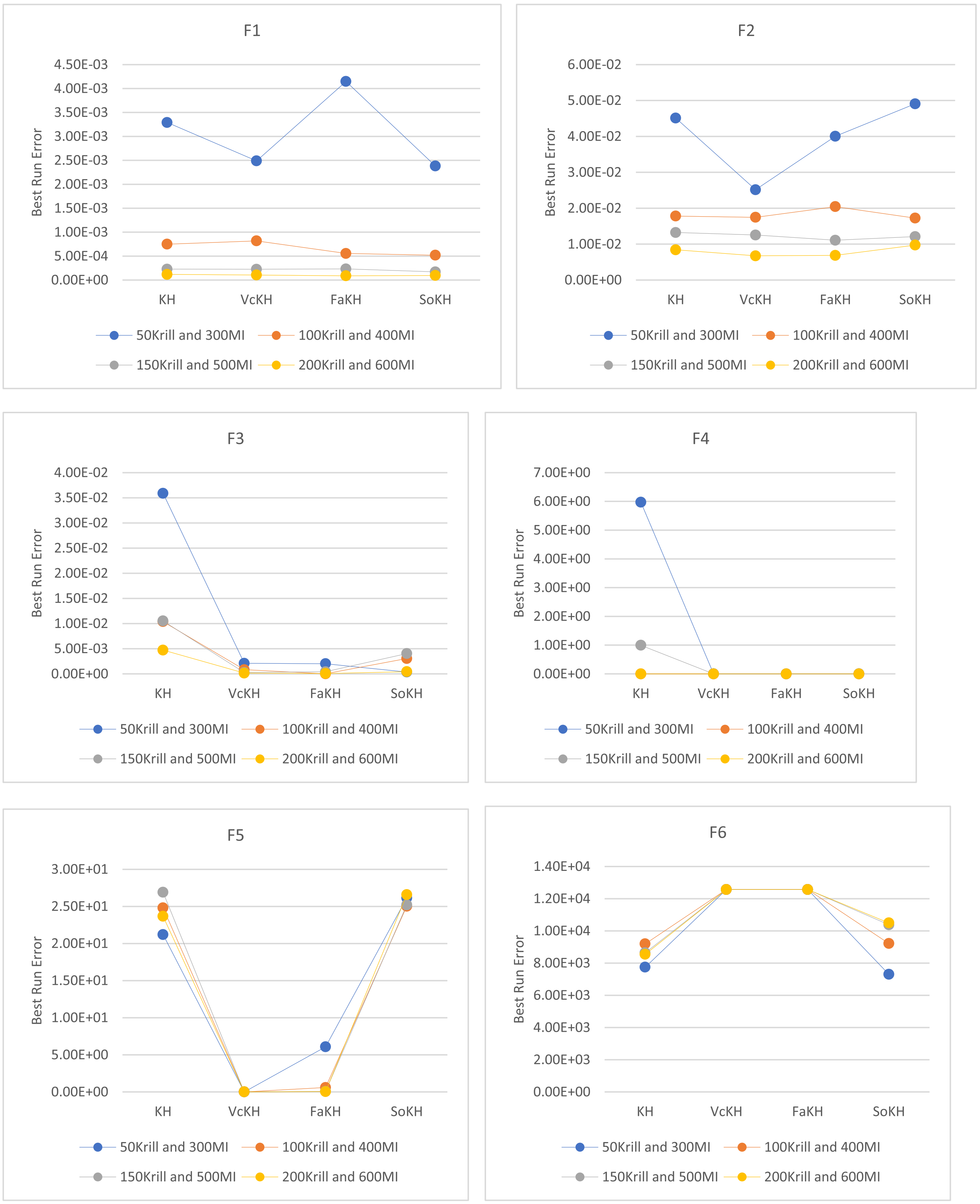
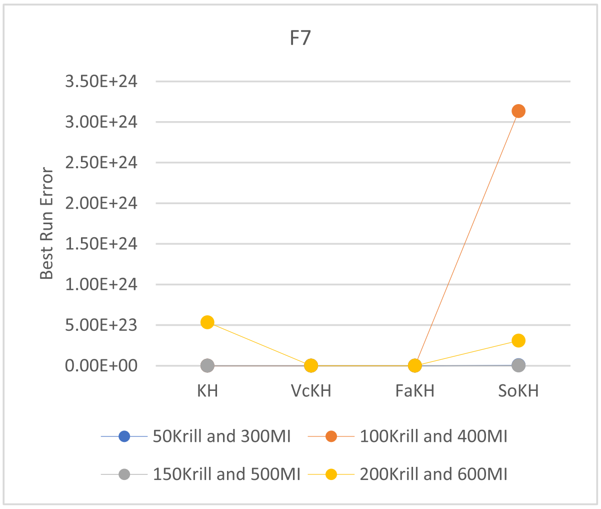
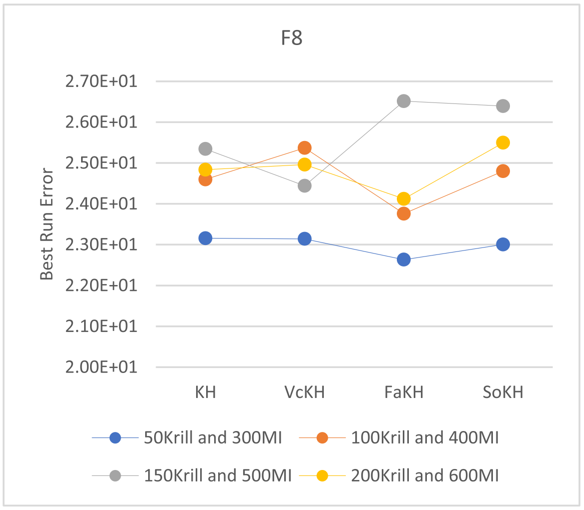
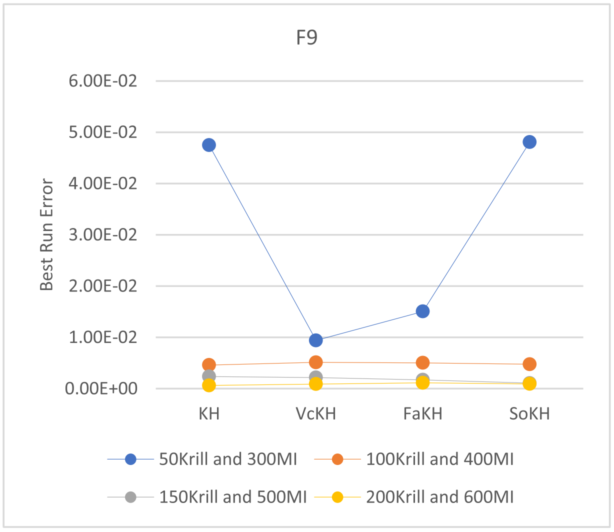
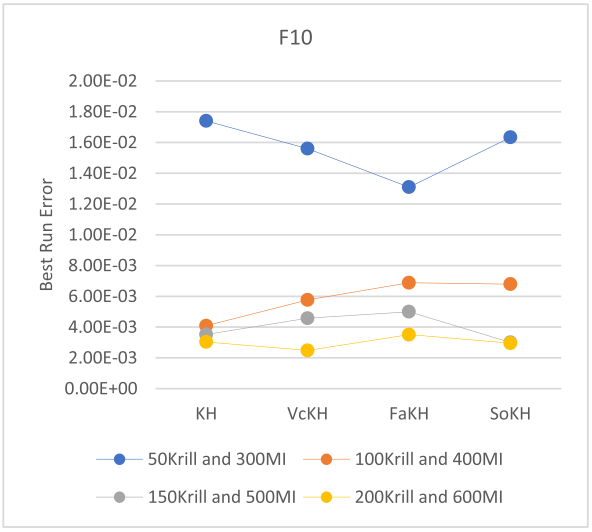


**FIGURE 8.** **Best-run error curves for 50-200 krill and maximum iteration 300-600.**









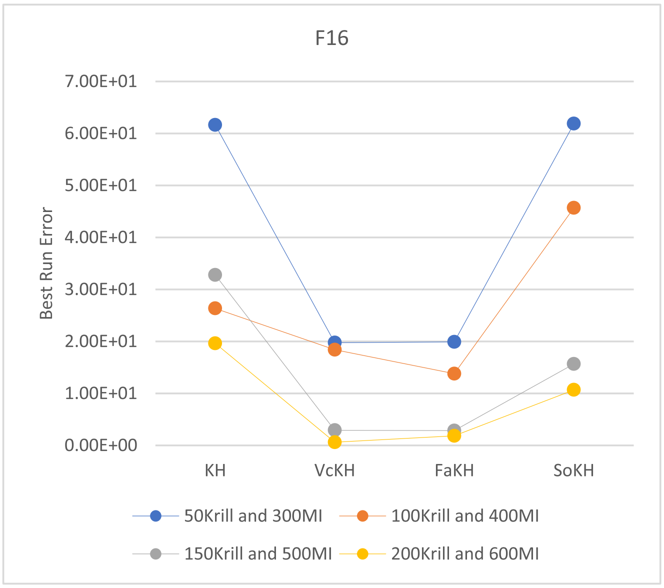


**FIGURE 8.** ***(Continued.)*** **Best-run error curves for 50-200 krill and maximum iteration 300-600.**

**TABLE 12.** Average rankings returned by the Friedman's non-parametric test for the 4 methods.

| Algorithm | Mean Rank (10D, 25NK) | Mean Rank (30D, 25NK) | Mean Rank (30D, 50NK) | Mean Rank (30D, 200NK) |
|---|---|---|---|---|
| KH | 2.85 | 3.73 | 2.82 | 3.10 |
| VcKH | 2.45 | 1.41 | 1.45 | 1.85 |
| FaKH | 1.98 | 2.32 | 2.95 | 2.15 |
| SoKH | 2.73 | 2.55 | 2.77 | 2.90 |

*D=denotes dimension, NK = population size of KH algorithm

Figure 7 gives the evaluation of the best-run error curve for the experiment case 1; clearly, we see that in most cases, especially when the dimension is high, the improved krill herd algorithms outperform the basic krill herd algorithm. The algorithms perform more or less the same when the dimension is low. It can also be seen from Figure 1 that the SoKH performs better than do the other algorithms for higher dimensions, which is in line with results presented in [2]. The FaKH and VcKH algorithms have relatively similar performances, and KH performs poorly as the dimension grows beyond 10. This is plausible because as the search space becomes larger, the randomly distributed krill are not uniformly distributed across the search space, contrasting with the case when the krill are distributed using the low-discrepancy sequences.

Figure 8 gives the evaluation of the best run error curve for experiment case 2, which shows the FaKH and VcKH algorithms performing better than do the remaining algorithms. This is plausible because, in the study presented in [3], it was reported that Faure and Van der Corput sequences are useful only when the dimension is not more than 30. All the algorithms showed improvements as we tuned the number of krill herd and maximum iterations; this can be attributed to the search space remaining the same, while the number of uniformly distributed krill increased, thereby covering more space and leading to better results. However, this does not guarantee that continued increments in the krill population will lead to a better solution; this is seen in the case of SoKH whose performance deteriorated as we increased the population size.

### C. STATISTICAL ANALYSIS

In order to validate the initial claim of the superior performances of the three proposed algorithms, a further statistical analysis test was carried out using the computed average solution as the response variable. The one-way ANOVA test was initially applied, and the three main assumptions, namely, normality, homoscedasticity and independence, were first tested. It was discovered that the normality test was not satisfied. Therefore, the Friedman non-parametric test was performed. For the experimentation with small dimension dataset of 10-dimension, the Friedman test revealed that there was no statistically significant ($p = 0.126$) difference in the algorithms performance whilst running: $\chi^2$ (3, N = 20) = 5.370. However, there was a noticeable statistically significant difference among the performances of the three algorithms as the dimension of the test problem increases. Specifically, with the 30-dimension test problems, the VcKH showed outstanding performance whilst running, $\chi^2(3) = 18.193$, $p = 0.001$. Further, Table 12 presents the average rankings returned by Friedman's non-parametric test for the traditional KH and the three new methods. In addition, it provides insight as to how each method performs overall considered problem instances [71].

## V. CONCLUSION AND FUTURE WORK

This article has shown that it is worth using the quasi-random topology sequence or low-discrepancy sequences instead of a random number generator for the initialization of a krill herd. This is because the low-discrepancy sequences are more uniformly distributed across the search space than are the random numbers. Hence, optimal results are obtained, especially for large dimensions. We used three low-discrepancy sequences (Faure, Sobol, and Van der Caput) to initialize the krill population in the basic KH algorithm. Most existing improvements on the KH algorithm have not focused on the initialization of the krill, and therefore, our improvements can easily be incorporated into these earlier variations.

Our experimental results showed significant improvements in the performance of the modified KH algorithm, particularly for high dimensioned test problems, where our proposed improvements outperformed the basic KH for all functions evaluated. The results for low dimensions showed some variations, as the performance of our proposed improvements was much the same as for the basic KH algorithm. Specifically, the proposed improved mechanisms outperformed the basic KH in most cases when the dimension of the problem instance is set at 20. When the number of krill is set at 25, on the one hand SoKH is seen to be the best-performing algorithm for high-dimension test problems. This is in line with the findings in [2], [5], where the PSO was initialized using a low-discrepancy sequence. On the other hand, the performance of VcKH and FaKH are interchangeable for dimension set at 20. Furthermore, as we vary the number of krill (for example 50, 100, 150, and 200), we saw the performance of SoKH deteriorate, whereas FaKH and VcKH continue to outperform the other algorithms. This could mean that SoKH is sensitive to the population size.

In all cases, there were significant improvements in the performance of KH when initialized with low-discrepancy sequences. Although our work is limited to initialization, other motions of the krill such as physical diffusion, which is random motion adopted by the krill, could be influenced using low-discrepancy sequences or by other metaheuristic algorithms. Therefore, this mechanism of diffusion could be explored in future work. In addition, we plan to extend the initialization schemes used here with other probability distributions such as the exponential distribution, beta distribution, and so on, to improve the performance of other new-generation metaheuristics algorithms.

## DECLARATION OF COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this article.

## REFERENCES

[1] C. Grosan, A. Abraham, and M. Nicoara, "Search optimization using hybrid particle sub-swarms and evolutionary algorithms," *Int. J. Simul. Syst., Sci. Technol.*, vol. 6, no. 10, pp. 60–79, 2005.

[2] M. Pant, R. Thangaraj, C. Grosan, and A. Abraham, "Improved particle swarm optimization with low-discrepancy sequences," in *Proc. IEEE Congr. Evol. Comput., IEEE World Congr. Comput. Intell.*, Jun. 2008, pp. 3011–3018.

[3] J. E. Gentle, *Random Number Generation and Monte Carlo Methods*. New York, NY, USA: Springer, 2006.

[4] S. Kimura and K. Matsumura, "Genetic algorithms using low-discrepancy sequences," in *Proc. 7th Annu. Conf. Genet. Evol. Comput. (GECCO)*, 2005, pp. 1341–1346.

[5] N. Q. Uy, N. X. Hoai, R. McKay, and P. M. Tuan, "Initialising PSO with randomised low-discrepancy sequences: The comparative results," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2007, pp. 1985–1992.

[6] K. E. Parsopoulos and M. N. Vrahatis, "Particle swarm optimizer in noisy and continuously changing environments," *Methods*, vol. 5, no. 6, p. 23, 2001.

[7] R. Brits, A. P. Engelbrecht, and F. Van den Bergh, "A niching particle swarm optimizer," in *Proc. 4th Asia–Pacific Conf. Simulated Evol. Learn.*, vol. 2, Nov. 2020, pp. 692–696.

[8] R. Brits, A. P. Engelbrecht, and F. van den Bergh, "Solving systems of unconstrained equations using particle swarm optimization," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 3, Oct. 2002, p. 6.

[9] G.-G. Wang, A. H. Gandomi, A. H. Alavi, and D. Gong, "A comprehensive review of krill herd algorithm: Variants, hybrids and applications," *Artif. Intell. Rev.*, vol. 51, no. 1, pp. 119–148, Jan. 2019.

[10] G.-G. Wang, L. Guo, A. H. Gandomi, G.-S. Hao, and H. Wang, "Chaotic krill herd algorithm," *Inf. Sci.*, vol. 274, pp. 17–34, Aug. 2014.

[11] G.-G. Wang, S. Deb, A. H. Gandomi, and A. H. Alavi, "Opposition-based krill herd algorithm with cauchy mutation and position clamping," *Neurocomputing*, vol. 177, pp. 147–157, Feb. 2016.

[12] G. Wang, L. Guo, A. H. Gandomi, L. Cao, A. H. Alavi, H. Duan, and J. Li, "Lévy-flight krill herd algorithm," *Math. Problems Eng.*, vol. 2013, pp. 1–14, Feb. 2013.

[13] G.-G. Wang, A. H. Gandomi, A. H. Alavi, and S. Deb, "A multi-stage krill herd algorithm for global numerical optimization," *Int. J. Artif. Intell. Tools*, vol. 25, no. 2, Apr. 2016, Art. no. 1550030.

[14] G. G. Wang, A. H. Gandomi, X. S. Yang, and A. H. Alavi, "A new hybrid method based on krill herd and cuckoo search for global optimization tasks," *Int. J. Bio-Inspired Comput.*, vol. 8, no. 5, pp. 286–299, 2016.

[15] L. M. Abualigah, A. T. Khader, and E. S. Hanandeh, "A hybrid strategy for krill herd algorithm with harmony search algorithm to improve the data clustering?" *Intell. Decis. Technol.*, vol. 12, no. 1, pp. 3–14, 2018.

[16] G.-G. Wang, A. H. Gandomi, and A. H. Alavi, "Stud krill herd algorithm," *Neurocomputing*, vol. 128, pp. 363–370, Mar. 2014.

[17] G.-G. Wang, A. H. Gandomi, and A. H. Alavi, "An effective krill herd algorithm with migration operator in biogeography-based optimization," *Appl. Math. Model.*, vol. 38, nos. 9–10, pp. 2454–2462, May 2014.

[18] G.-G. Wang, A. H. Gandomi, A. H. Alavi, and G.-S. Hao, "Hybrid krill herd algorithm with differential evolution for global numerical optimization," *Neural Comput. Appl.*, vol. 25, no. 2, pp. 297–308, Aug. 2014.

[19] G. G. Wang, S. Deb, and S. M. Thampi, "A discrete krill herd method with multilayer coding strategy for flexible job-shop scheduling problem," in *Intelligent Systems Technologies and Applications*. Cham, Switzerland: Springer, 2016, pp. 201–215.

[20] D. Rodrigues, L. A. M. Pereira, J. P. Papa, and S. A. T. Weber, "A binary krill herd approach for feature selection," in *Proc. 22nd Int. Conf. Pattern Recognit.*, Aug. 2014, pp. 1407–1412.

[21] E. Fattahi, M. Bidar, and H. R. Kanan, "Fuzzy krill herd optimization algorithm," in *Proc. 1st Int. Conf. Netw. Soft Comput. (ICNSC)*, Aug. 2014, pp. 423–426.

[22] H. V. H. Ayala, E. H. V. Segundo, V. C. Mariani, and L. dos S. Coelho, "Multiobjective krill herd algorithm for electromagnetic optimization," *IEEE Trans. Magn.*, vol. 52, no. 3, pp. 1–4, Mar. 2016.

[23] P. A. Kowalski and S. Łukasik, "Training neural networks with krill herd algorithm," *Neural Process. Lett.*, vol. 44, no. 1, pp. 5–17, Aug. 2016.

[24] H. Pulluri, R. Naresh, and V. Sharma, "A solution network based on stud krill herd algorithm for optimal power flow problems," *Soft Comput.*, vol. 22, no. 1, pp. 159–176, Jan. 2018.

[25] S. Dutta, P. Mukhopadhyay, P. K. Roy, and D. Nandi, "Unified power flow controller based reactive power dispatch using oppositional krill herd algorithm," *Int. J. Electr. Power Energy Syst.*, vol. 80, pp. 10–25, Sep. 2016.

[26] H. V. H. Ayala, E. H. V. Segundo, V. C. Mariani, and L. dos S. Coelho, "Multiobjective krill herd algorithm for electromagnetic optimization," *IEEE Trans. Magn.*, vol. 52, no. 3, pp. 1–4, Mar. 2016.

[27] A. H. Gandomi and A. H. Alavi, "An introduction of krill herd algorithm for engineering optimization," *J. Civil Eng. Manage.*, vol. 22, no. 3, pp. 302–310, Aug. 2015.

[28] J. Van Der Coput, "Verteilungsfunktionen I & II," in *Proc. Nederl. Akad. Wetensch.*, vol. 38, 1935, pp. 1058–1066.

[29] A. H. Gandomi and A. H. Alavi, "Krill herd: A new bio-inspired optimization algorithm," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 17, no. 12, pp. 4831–4845, Dec. 2012.

[30] K. E. Parsopoulos and M. N. Vrahatis, "Initializing the particle swarm optimizer using the nonlinear simplex method," in *Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, A. Grmela and N. Mastorakis, Eds. Interlaken, Switzerland: WSEAS Press, 2002, pp. 216–221.

[31] M. Richards and D. Ventura, "Choosing a starting configuration for particle swarm optimization," in *Proc. IEEE Int. Joint. Conf. Neural Netw.*, vol. 3, Jul. 2004, pp. 2309–2312.

[32] X. Wang and F. J. Hickernell, "Randomized halton sequences," *Math. Comput. Model.*, vol. 32, nos. 7–8, pp. 887–899, Oct. 2000.

[33] E. I. Atanassov, "Efficient CPU-specific algorithm for generating the generalized Faure sequences," in *Proc. Int. Conf. Large-Scale Sci. Comput.* Berlin, Germany: Springer, Jun. 2003, pp. 121–127.

[34] A. P. Engelbretch, *Fundamentals of Computational Swarm Intelligence*. Hoboken, NJ, USA: Wiley, 2005, pp. 5–129.

[35] E. Cantú-Paz, "On random numbers and the performance of genetic algorithms," *Comput. Sci. Preprint Arch.*, vol. 2002, no. 10, pp. 203–210, 2002.

[36] J. M. Daida, S. Ross, J. McClain, D. Ampy, and M. Holczer, "Challenges with verification, repeatability, and meaningful comparisons in genetic programming," in *Proc. Genet. Program., 2nd Annu. Conf.* San Mateo, CA, USA: Morgan Kaufmann, 1997, pp. 64–69.

[37] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.

[38] D. E. Knuth, *The Art of Computer Programming*, vol. 3. London, U.K.: Pearson, 1997.

[39] W. J. Morokoff and R. E. Caflisch, "Quasi-random sequences and their discrepancies," *SIAM J. Sci. Comput.*, vol. 15, no. 6, pp. 1251–1279, Nov. 1994.

[40] E. I. Atanassov, "A new efficient algorithm for generating the scrambled Sobol'sequence," in *Proc. Int. Conf. Numer. Methods Appl.* Berlin, Germany: Springer, 2002, pp. 83–90.

[41] P. Bratley and B. L. Fox, "Algorithm 659: Implementing Sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 14, no. 1, pp. 88–100, 1988.

[42] S. Joe and F. Y. Kuo, "Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 29, no. 1, pp. 49–57, Mar. 2003.

[43] D. K. Gehlhaar, "Tunig evolutionary programming for conformationally flexible molecular docking," in *Proc. 5th Annu. Conf. Evol. Program.*, 1996, pp. 419–429.

[44] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.

[45] A. E. Ezugwu, O. J. Adeleke, A. A. Akinyelu, and S. Viriri, "A conceptual comparison of several Metaheuristic algorithms on continuous optimisation problems," *Neural Comput. Appl.*, vol. 32, no. 10, pp. 6207–6251, May 2020.

[46] V. Picheny, T. Wagner, and D. Ginsbourger, "A benchmark of kriging-based infill criteria for noisy optimization," *Struct. Multidisciplinary Optim.*, vol. 48, no. 3, pp. 607–626, Sep. 2013.

[47] W. Qiao and Z. Yang, "Solving large-scale function optimization problem by using a new Metaheuristic algorithm based on quantum dolphin swarm algorithm," *IEEE Access*, vol. 7, pp. 138972–138989, 2019.

[48] P. A. Kowalski and S. Łukasik, "Experimental study of selected parameters of the krill herd algorithm," in *Intelligent System*. Cham, Switzerland: Springer, 2015, pp. 473–485.

[49] A. Søren and W. G. Peter, *Stochastic Simulation: Algorithms and Analysis*. Springer, 2007, p. 476.

[50] S. Kondamadugula and S. R. Naidu, "Accelerated evolutionary algorithms with parameterimportance based population initialization for variation-aware analog yield optimization," in *Proc. IEEE 59th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Oct. 2016, pp. 1–4.

[51] H. Maaranen, K. Miettinen, and M. M. Mäkelä, "Quasi-random initial population for genetic algorithms," *Comput. Math. With Appl.*, vol. 47, no. 12, pp. 1885–1895, Jun. 2004.

[52] L. D. S. Coelho and V. C. Mariani, "Use of chaotic sequences in a biologically inspired algorithm for engineering design optimization," *Expert Syst. Appl.*, vol. 34, no. 3, pp. 1905–1913, Apr. 2008.

[53] C. Li, X. Chu, Y. Chen, and L. Xing, "A knowledge-based initialization technique of genetic algorithm for the travelling salesman problem," in *Proc. 11th Int. Conf. Natural Comput. (ICNC)*, Aug. 2015, pp. 188–193.

[54] Q. Li, S.-Y. Liu, and X.-S. Yang, "Neighborhood information-based probabilistic algorithm for network disintegration," *Expert Syst. Appl.*, vol. 139, Jan. 2020, Art. no. 112853.

[55] Q. Li, S.-Y. Liu, and X.-S. Yang, "Influence of initialization on the performance of Metaheuristic optimizers," *Appl. Soft Comput.*, vol. 91, Jun. 2020, Art. no. 106193.

[56] M. Bhowmik and P. Malathi, "Spectrum sensing in cognitive radio using actor–critic neural network with krill herd-whale optimization algorithm," *Wireless Pers. Commun.*, vol. 105, no. 1, pp. 335–354, Mar. 2019.

[57] P. G. Asteris, S. Nozhati, M. Nikoo, L. Cavaleri, and M. Nikoo, "Krill herd algorithm-based neural network in structural seismic reliability evaluation," *Mech. Adv. Mater. Struct.*, vol. 26, no. 13, pp. 1146–1153, Jul. 2019.

[58] D. S. Dharrao and N. J. Uke, "Fractional Krill–Lion algorithm based actor critic neural network for face recognition in real time surveillance videos," *Int. J. Comput. Intell. Appl.*, vol. 18, no. 02, Jun. 2019, Art. no. 1950011.

[59] S. Balamurugan and M. L. Sundar, "Krill herd-based optimal neural network for analyzing safety and quality performance at construction site," *Int. J. Rapid Manuf.*, vol. 8, no. 4, pp. 345–363, 2019.

[60] A. Ashrafzadeh, M. A. Ghorbani, S. M. Biazar, and Z. M. Yaseen, "Evaporation process modelling over northern iran: Application of an integrative data-intelligence model with the krill herd optimization algorithm," *Hydrological Sci. J.*, vol. 64, no. 15, pp. 1843–1856, Nov. 2019.

[61] L. M. Abualigah, A. T. Khader, and E. S. Hanandeh, "Modified krill herd algorithm for global numerical optimization problems," in *Advances in Nature-Inspired Computing and Applications*. Cham, Switzerland: Springer, 2019, pp. 205–221.

[62] Z. Zhang, S. Ding, and Y. Sun, "A support vector regression model hybridized with chaotic krill herd algorithm and empirical mode decomposition for regression task," *Neurocomputing*, vol. 410, pp. 185–201, Oct. 2020.

[63] Z. M. Ali, N. Vu Quynh, S. Dadfar, and H. Nakamura, "Variable step size perturb and observe MPPT controller by applying $\theta$-modified krill herd algorithm-sliding mode controller under partially shaded conditions," *J. Cleaner Prod.*, vol. 271, Oct. 2020, Art. no. 122243.

[64] P. A. Kowalski, S. Łukasik, M. Charytanowicz, and P. Kulczycki, "Nature inspired clustering-use cases of krill herd algorithm and flower pollination algorithm," in *Interactions Between Computational Intelligence and Mathematics Part 2*. Cham, Switzerland: Springer, 2019, pp. 83–98.

[65] M. Akbari and H. Izadkhah, "Hybrid of genetic algorithm and krill herd for software clustering problem," in *Proc. 5th Conf. Knowl. Based Eng. Innov. (KBEI)*, Feb. 2019, pp. 565–570.

[66] A. Khelifi, B. Bentouati, S. Chettih, and R. A. El-Sehiemy, "A novel hybrid method based on krill herd and cuckoo search for optimal power flow problem," *Int. J. Image, Graph. Signal Process.*, vol. 11, no. 9, pp. 1–17, Sep. 2019.

[67] A. Khelifi, B. Bentouati, S. Chettih, and R. El-Sehiemy, "A novel hybrid krill herd algorithm for improving the performance of electric power systems operation," *Model., Meas. Control A*, vol. 92, nos. 2–4, pp. 86–96, Dec. 2019.

[68] P. K. Adhvaryyu and S. Adhvaryyu, "Static optimal load flow of combined heat and power system with valve point effect and prohibited operating zones using Krill Herd algorithm," *Energy Syst.*, vol. 11, no. 1, pp. 1–24, 2020.

[69] D. Saravanan, S. Janakiraman, K. Chandraprabha, T. Kalaipriyan, R. S. Raghav, and S. Venkatesan, "Augmented powell-based krill herd optimization for roadside unit deployment in vehicular ad hoc networks," *J. Test. Eval.*, vol. 47, no. 6, pp. 4108–4127, 2019.

[70] A. L. Bolaji, M. A. Al-Betar, M. A. Awadallah, A. T. Khader, and L. M. Abualigah, "A comprehensive review: Krill herd algorithm (KH) and its applications," *Appl. Soft Comput.*, vol. 49, pp. 437–446, Dec. 2016.

[71] A. E. Ezugwu, "Enhanced symbiotic organisms search algorithm for unrelated parallel machines manufacturing scheduling with setup times," *Knowl.-Based Syst.*, vol. 172, pp. 15–32, May 2019.

**OVRE JEFFREY AGUSHAKA** (Member, IEEE) received the B.Sc. degree in mathematics with computer science and the M.Sc. degree in computer science from Ahmadu Bello University, Zaria, Nigeria. He is currently pursuing the Ph.D. degree in computer science with the School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban, South Africa.

**ABSALOM EL-SHAMIR EZUGWU** (Member, IEEE) received the B.Sc. degree in mathematics with computer science and the M.Sc. and Ph.D. degrees in computer science from Ahmadu Bello University, Zaria, Nigeria. He is currently a Senior Lecturer with the School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal, Durban, South Africa. His main research interests include parallel algorithms design in cloud and grid computing environments, artificial intelligence with specific interest to computational intelligence, and metaheuristics solutions to real-world global optimization problems. He has published articles relevant to his research interest in internationally refereed journals and edited books, conference proceedings, and local journals. He is a member of IAENG and ORSSA.