Nguyen Quang Uy, Nguyen Xuan Hoai, RI McKay, and Pham Minh Tuan

Initialising PSO with Randomized Low-Discrepancy Sequences: The Comparative Results

Abstract— **In this paper, we investigate the use of some wel-known randomized low-discrepancy sequences (Halton, Sobol, and Faure sequences) for initialising particle swarms. We experimented with the standard global-best particle swarm algorithm for function optimisation on some benchmark problems, using randomized low-discrepancy sequences for initialisation, and the results were compared with the same particle swarm algorithm using uniform initialisation with a pseudo-random generator. The results show that~~,~~ the former initialisation method could help the particle swarm algorithm improve its performance over the latter on the problems tried. Furthermore the comparisons also indicate that the use of different randomized low-discrepancy sequences in the initialisation phase could bring different effects on the performance of PSO.**

I. Introduction

Almost all evolutionary algorithms (EAs) proposed so far employ some sort~~s~~ of random decision making. However computers cannot provide truly random numbers. Consequently, like many randomized algorithms, most EAs use pseudo-random number generators for their random decision making. Similarly, for the implementation of Monte Carlo Methods on computers, pseudo-random generators have been used to simulate the uniform distribution [6]. The performance of the Monte Carlo Methods is known to be heavily dependant on the quality of the pseudo-random generators. Likewise, several studies in evolutionary computation (EC) have suggested that the use of different pseudo-random generators can have significant effects on performance [5, 7, 15, 16].

However, it is reported in the Monte Carlo Methods literature that pseudo-random number generators cannot achieve optimal discrepancy (i.e. small deviation from the uniform distribution) [17, 19]. Consequently, researchers have studied alternative ways to generate low-discrepancy sequences for stratified sampling in Monte Carlo Methods. The best-known low-discrepancy sequences are deterministic (known as 'quasi-random'), and can achieve near-optimal discrepancy. Some famous such sequences include Halton, Sobol, Faure, Niederreiter sequences [10, 19]. The class of Monte-Carlo Methods using low-discrepancy sequences for stratified sampling is now known as 'Quasi Monte Carlo Methods'.

Inspired by this transition from Monte Carlo Methods to Quasi Monte Carlo Methods, it is interesting to see whether low-discrepancy sequences (in particular their scrambled/-randomized versions) are useful for EAs. To the best of our knowledge, there has only been a very limited ~~of~~ number of studies in the EA literature addressing this issue: [4, 13, 20, 21].—

In this paper, we investigate the use of three randomized low-discrepancy sequences (Halton, Faure, Sobol sequences) for initialising particle swarms [8], a new emerging nature-inspired meta~~e~~ heuristic technique in the field of EAs and Swarm Intelligence. The standard global-best particle swarm algorithm [9] for function optimisation using randomized low-discrepancy sequences was applied to some benchmark problems, and the results were compared with the same particle swarm algorithm using the more common uniform initialisation (with a pseudo-random generator). The paper is organized as follows. In the next section, we give a brief introduction to some related work in the literature. Section 3 contains some background on pseudo-random generators, low-discrepancy sequences, and the version of particle swarm we used for the experiments presented in section 4. The paper concludes with section 5, where some future proposals for extending the work in this paper are put forward.

II. Related Work

Our work is much inspired by [13], where the randomized Halton sequence was used to generate stratified (uniform) samples for a real-coded GAs. In that work, Kimura and Matsumura showed that the real-coded GA could benefit from low discrepancy sequences as a more uniform (than pseuro-random generators) way of initialising the GA population. They discovered that the performance of the real-coded GA using randomized Halton sequence is superior to the performance of the same GA using a pseudo-random generator for population generation. Our work extends this approach to the field of particle swarm optimization, where we are not aware of any similar preceding work.

Particle Swarm Optimization (PSO) [9] is a newly emerging computational methodology in natural computation. A PSO algorithm maintains a swarm of particles, where each particle represents a potential solution. This swarm of particles 'flows' in the multi-dimensional solution space according to some physical or nature based rules (such as the rules observed in the behaviour of a flock of birds) [9]. Iterations between particles through some mechanisms of individual and social learning help to attract the particle swarm towards the areas of optimal solutions. Since it was first introduced in [8, 12], PSO has quickly become a promising and on-going area of research in natural computation, with many applications and extensions [9, 11, 24].

Nguyen Quang Uy, Nguyen Xuan Hoai, and Pham Minh Tuan are with the NC research group, Department of IT, Military Technical Academy, 100 Hoang Quoc Viet St., Hanoi, Vietnam (corresponding author's e-mail: hoainx@lqdtu.edu.vn).

R.I. McKay is with the Structural Complexity Laboratory, Seoul National University, Korea (e-mail: **rim@cse.snu.ac.kr**).

One of the important components in a particle swarm algorithm is the initialisation of particle positions. It is suggested that the performance of PSO is heavily dependent on the initial positions of the particles [9]. Moreover, in the absence of knowledge about the search/solution space, it is desirable that the particles are initialised as widely spread in the search/solution space as possible [9]. Consequently, there have been a number of attempts to propose different methods for PSO initialisation. In the first implementations of PSO, the particles were initialised uniformly at random (assuming the use of pseudo-random generators). This initialisation strategy became the most popular strategy in PSO subsequently [9].

In [20] and [4], Sobol and Faure low-discrepancy sequences were employed to initialise the swarm of particles. We are unsure of the details, as there is limited discussion of the details for the implementation. In particular, it is not clear to us whether they used deterministic or randomised low-discrepancy sequences, as the references they cited did not contain implementations of scrambled (randomized) Sobol and Faure sequences. Randomization of low-discrepancy sequences is important if they are to be used for multi-start (or multiple run) random (heuristic) search algorithms. The underlying algorithms generate deterministic point sequences designed to fill up the search space (usually the unit cube). Thus in their raw form, repeated runs will result in the same output.

To the best of our knowledge, there has been no previous detailed investigation of the effect of randomized low-discrepancy sequences on the performance of PSO, except our preliminary work [18]. In [18], we did some experiments on using the randomized Halton sequence for initialising PSO and compared the performance with the PSO initialised by a pseudo-random generator. The results were ~~less convincing~~statistically weak due to the huge variations in performance of different runs, which was a direct consequence of our experimental~~s~~ settings. This paper extends that preliminary work by changing the experiment setting (discussed in Section 4) and implements new PSO initialisation methods by using two more randomized low-discrepancy sequences (i.e Sobol and Faure sequences).

Low-discrepancy uniform initialisation of PSO~~,~~ is not the only strand of work on improved initialisation for PSO. In [21], Parsopoulos and Vrahatis used the nonlinear simplex method to initialise PSO. The particles are then moved towards better solutions by local search. So this method of initialisation is based on the exploitation of the search space to start with good solutions. In [23], an initial particle is placed at the centre of the search space, and from there the rest of the particles are spread over the search space through clustering of the search space. This initialisation method could, however, be relatively biased, as many benchmark objective functions for PSO have optima at the centre of the search space., so that performance on benchmark functions could be difficult to transfer to real-world functions.

III. BACKGROUNDS

In this section, we first give a brief summary of pseudo-random generation and low-discrepancy sequences, emphasising the randomized low-discrepancy sequences which we use in our PSO initialisation.

A. Random Number Generation

Modern computers are deterministic in nature. Therefore, it seems perverse to ask a computer to generate random number. Nevertheless, random numbers are essential for randomized algorithms, an important class of problem-solving methodologies using random decision making in their processes. The quality of random numbers (i.e how truly random they are) is crucial to almost all randomized computation methods, such as Monte Carlo Methods and EAs, as evidenced in the literature [5, 7, 9, 15, 16]. One of the measures for the quality of pseudo-random generators is uniformity. Uniformity is usually evaluated by the discrepancy (i.e. the deviation from the true uniform distribution). For a point set $P=\{x_1,x_2,...,x_N\}$ in $[0..1]^s$ the (star) discrepancy of P is computed as [19, 25]:

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

where $\mathbf{u}=(u_1,u_2,..u_s)$, J is the hyper-rectangle defined by $[0..u_i]$ $(i=1,2...,s)$, $A(J,P_N)$ is the number of points inside J, and V(J) is the volume of J. For other tests of the goodness of ~~a~~ pseudo-random generators, we recommend [14] as a good and complete source of references.

There are two main streams of algorithms for generating pseudo-random numbers, namely~~,~~ the linear congruential method and the feedback shift register method [10, 14]. Of the two, the linear congruential method is far more popular. Therefore in this work, we choose the linear congruential method to generate pseudo-random numbers. The most popular and widely used linear congruential pseudo-random generators are based on the Lehmer generator (or Lehmer sequence). The form of the generator is:

$x_i=(ax_i+c) \bmod m;\ 0 \leq x_i<m$

In practice, c is usually set as 0, and the resultant pseudo-random generators are called multiplicative congruential generators. The quality of that kind of pseudo-random generators is very much dependent on the choices of a and m [10, 14]. The implementation of a multiplicative congruential pseudo-random generator in this paper is taken from [22], as it has been used and tested for quite some time both by our~~-~~selves and the wider research community.

B. Low-Discrepancy Sequences

In [17], it was shown that uniform pseudo-random number sequences have discrepancy of order $(\log(\log(N)))^{1/2}$ and thus do not achieve the lowest possible discrepancy. Subsequently, researchers have proposed an alternative way of generating ‘quasi-random’ numbers through the use of low discrepancy sequences. Low discrepancy sequences are designed to be deterministic (less random than pseudo-random numbers) but more uniform ~~(stratified)~~ than pseudo-random numbers. Their discrepancies have been shown to be optimal, of order $(\log(N))^s/N$ [10, 14, 25]. A number of such quasi-random sequences have been proposed: Halton, Sobol, Faure, and Niederreiter, to name but a few. They have been extensively used in generating stratified samples for Quasi-Monte Carlo Methods.

In this paper, we use the randomized Halton, Sobol, and Faure sequences. The randomiz~~ation~~ed (scramble~~s~~ds) versions of these sequences are based on [1-3, 10, 25]. Furthermore, the randomization, proven to preserve discrepancy [25], supports multiple runs (multi-start) of randomized algorithms/heuristics (such as PSO), which in turn facilitates comparison with pseudo-random numbers.

B.1. The Randomized Halton Sequence

The Halton sequence is an extension of the van der Corput sequence (from 1 dimension to n dimensions). The van der Corput sequence in base b is a one dimensional low discrepancy sequence defined as follows [10, 25]:

For an integer b $\geq$ 2, we set $Z_b=\{0, 1, ...b-1\}$ then every integer n $\geq$ 0 has a unique digit expansion in base b as:

$$n = \sum_{j=0}^{m} a_j b^j$$

where $a_j \in Z_b$ for $j \geq 0$, and $m = \lfloor \log_b n \rfloor$. We define $\varphi_b(n)$ as the radical inverse function in base b for every b $\geq$ 2 as:

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

The Van de Corput sequence $S_b=\{t_0,t_1,...\}$ in base b is then defined as: $t_n = \varphi_b(n)$.

The van de Corput sequence is a low discrepancy sequence in one dimensional space. In order to generate quasi-random numbers in multi-dimensional space, some extensions are needed. One of the extensions of the van de Corput sequence is the Halton sequence [10, 25], defined in the s-dimensional space as follows:

$x_n=(\varphi_{b1}(n),\varphi_{b2}(n), ..., \varphi_{bs}(n))$

where b_1, b_2, ..., b_s are integers that are greater than one and pair-wise co-prime. In practice (and in our implementation), the bases are usually chosen as the first s primes.

As with other low-discrepancy sequences, the Halton sequence is deterministic. Thus it is not appropriate for our purpose in undertaking multiple runs (multiple starts) of PSO and comparing them with PSO using pseudo-random numbers for initialisation. Therefore, in this paper, we use the randomized Halton sequence. The randomization is created by adding Gaussian noise~~s~~ (with a mean of zero ~~in mean~~ and standard deviation of 0.05 in this paper) to each coordinate~~s~~ of the sequence points[1].

B.2. The Randomized Faure Sequence

The Faure sequence [2, 10] is a permutation of the Halton sequence. Unlike the Halton sequence, it uses the same base for each dimension. The base *m* is the smallest prime number that is greater than or equal to the number of dimensions in the problem and not smaller than 2. Denote the *k*th point by

$$Z_k \equiv (c_1, c_2, \cdots, c_d)$$

The first component C_1 is the one-dimensional Halton sequence $\varphi_m(1)$, $\varphi_m(2)$, ... To generate Faure sequences, follow the following procedure.

If $C_n=b_0{}^{m-1}+b_1{}^{m-2}+...+ b_r{}^{m-(r+1)}$ then

$Cn-1=a_0{}^{m-1}+a_1{}^{m-2}+...+a_r{}^{m-(r+1)}$

where

$$b_j \equiv \sum_{i \geq j}^{r} \binom{i}{j} a_i \mod m$$

The randomization of the Faure sequence is then ~~created~~ done in the same way ~~with~~ as for the randomized Halton sequence.

B.3. The Randomized Sobol Sequence

The construction of the Sobol sequence [1, 3, 6, 10] uses linear recurrence relations over the finite field, F_2, where $F_2 = \{0\ 1\}$. Let the binary expansion of the non-negative integer n be given by $n= n_1 2^0+ n_2 2^1+....+n_w 2^{w-1}$. Then the the nth element of the jth dimension of the Sobol sequence, $x_n{}^{(j)}$, can be generated by:

$$x_{n}{}^{(j)}=n_1 v_1{}^{(j)} \oplus n_2 v_2{}^{(j)} \oplus .... \oplus n_w v_w{}^{(j)}$$

where $v_j{}^{(i)}$ is a binary fraction called the ith direction number in the jth dimension. These direction numbers are generated by the following q-term recurrence relation:

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

We have i>q, and the bit, a_i, comes from the coefficients of a degree-q primitive polynomial over F_2. Different primitive polynomials are used to generate the Sobol direction numbers in each different dimension.

~~In the same way~~As with Faure and Halton sequences, we obtain randomized Sobol sequence by adding some small random noises to their coordinates.

C.The Global-best PSO

Since the introduction of PSO, there have been a number of extensions and variations to the standard algorithm. However, to demonstrate the effects of using randomized low discrepancy sequences for initialisation, we chose the most basic and standard version of PSO, namely the global-best (glbest) PSO [9]. We believe that our methods could be readily extended to variants of the basic PSO algorithm. The glbest PSO is as follows [9] (page 95):

```
1. Initialise a swarm of particles (points) in the n-dimensional space.
2. Repeat
   For each particle i=1,....S.n_s do
   // Set the personal best position
    if f(S.x_i)<f(S.y_i) then
       S.y_i=S.x_i
    end
   //Set the global-best position
```

[1] We also tried~~did use~~ the form of randomization proposed in [25], and the results ~~of~~ for PSO ~~was~~ were similar on all the problem tried in this paper. We have done some experiments and the standard deviation of 0.05 seems to be the best choice for all the problems tried.

```
    if f(S.y_i) < f(S.y*) then
          S.y*=S.y_i
       end
       end
     For each particle i=1,..., S.n_s do
       Update the velocity of particle i;
       Update the position of particle i;
     end
    Until stopping criteria are met
```

where S.n$_s$ is the number of particles in the swarm, $\mathbf{x}_i$ is the current position of particle i, $\mathbf{y}_i$ is the best (measured by the objective function f) position that the particle i visited in the past, and $\mathbf{y}^*$ is the global-best position so far of the whole swarm. The value in dimension j of the velocity of particle i, v_{ij}, is updated in time sequence t as follows:

$$v_{ij}(t+1)= v_{ij}(t) + c_1r_{ij}(t)[y_{ij}(t)-x_{ij}(t)] + c_2r_{2j}(t)[y^*_j(t)-x_{ij}(t)]$$

where $x_{ij}(t)$ is the position of particle i in dimension j at time t, c_1 and c_2 are two positive acceleration constants used for scaling, and r_{1j} and r_{2j} are uniform random values in the range [0,1]. The position of a particle i, $x_i(t+1)$ is updated as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

IV. EXPERIMENTS AND RESULTS

To investigate the effect on performance of using different randomized low-discrepancy sequences for PSO initialisation, and to compare with uniform initialisation (using a pseudo-random generator), we implemented four versions of the glbest PSO algorithm given in the previous section, and applied them to some benchmark problems of continuous function optimization. In the first version (called U-PSO), the swarm particles in the first step are generated in uniform random manner using a pseudo-random generator. In the other three versions, the randomized Halton sequence (H-PSO), randomized Faure sequence (F-PSO), and randomized Sobol sequence (S-PSO) are used to generate the initial swarm. Otherwise, all four algorithms are identical. All algorithms use the same pseudo-random generator except in the first step.

D. The Test Functions

We chose the following benchmark continuous functions in n-dimensional space for optimization using U-PSO and SH-PSO:

f$_1$: Spherical function:

$$f(x) = \sum_{j=1}^{n} x_j^2, \quad x_j \in [-100, 100]$$

f$_2$: Hyper-elipsoid function:

$$f(x) = \sum_{j=1}^{n} j^2 x_j^2 \quad x_j \in [-10, 10]$$

f$_3$: Ackley function:

$$f(x) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{j=1}^{n} xj^2}} - e^{\frac{1}{n}\sum_{j=1}^{n}\cos(2\pi x_j)} + 20 + e,$$

x_j ⊠ [-30, 30]

f$_4$: Griewank function:

$$f(x) = 1 + \frac{1}{4000}\sum_{j=1}^{n} x_j^2 - \prod_{j=1}^{n}\cos\left(\frac{x_j}{\sqrt{j}}\right), \quad x_j \in [-300, 300]$$

f$_5$: Rastrigin function:

$$f(x) = \sum_{j=1}^{n} (x_j^2 - 10\cos(2\pi x_j) + 10, \quad x_j \in [-10, 10]$$

f$_6$: Rosenbrock function:

$$f(x) = \sum_{j=1}^{n/2} [100(x_{2j} - x_{2j-1}^2)^2 + (1 - x_{2j-1})^2], \quad x_j \in [-5, 5]$$

For all of these functions, the optimal value is 0.

E. Experiment Settings

To investigate the effects of using different swarm sizes and numbers of generations for the comparison between PSO using uniform initialisation (U-PSO) and PSO using randomized low discrepancy sequences for initialisation, for each system, we ran three sets of experiments. The experiments used a fixed budget of a maximum~~al~~ 100000 function evaluations. The swarm sizes (numbers of particles – S.n$_s$) in these three sets of runs were 50, 100, and 200, so that the maximal numbers of generations were 2000, 1000, and 500 respectively. For each of the test function, we set the number of dimensions to 10, 20, 30, and 40. For each combination of swarm size (number of generation), test function (from f$_1$ to f$_6$), and search dimension, 100 runs were allocated to each of the two algorithms, making a total of 28800 runs in all. To avoid the problems of huge performance variation due to some runs getting~~s~~ trapped in local optima, reported in our preliminary work [18], we distinguish two set of runs: "successful" and "unsuccessful" runs. A run is defined as "successful" if it ~~could~~ finds a solution with function value lower than a threshold (10^{-3} in these experiments), when such a solution is found, the run terminates and the number of function evaluations (or number of generation) so far is recorded. Those runs that are not "successful" are defined as "unsuccessful". We use the "successful" runs to estimate the rate of finding global optima by different algorithms and use the number of function evaluations to evaluate how fast they approach the optima (speed of convergence). For the "unsuccessful" runs we compare the quality of the best solutions they found until generation 100000~~th~~.

F. Results and Discussions

The detailed results are presented in Tables 1, 2 and 3 overleaf. From the results in the Tables we ~~could~~ can see that the different initialisation methods ~~could~~ can result in different overall ~~the~~ PSO performances. The best performances in a row are highlighted by using ~~the~~ boldface ~~format~~type.

The results clearly show that, regardless of chosen objective functions (column F), particle~~s~~ sizes, search space dimension (column Dim), S-PSO (PSO initialised with the randomized Sobol sequence) was the best among all the four algorithms. S-PSO consistently had ~~bigger~~ a higher rate of success (i.e it more often found ~~the~~ solutions that have function value smaller than the threshold before 100000 function evaluations - indicated in column SUC). Moreover, it found optimal solutions with much ~~less number of~~fewer function evaluations (i.e it converged faster – indicated in column EVAL). Even when S-PSO failed to find the optima after 100000 ~~number of~~ function evaluations, the quality of the solutions it found were much higher than U-PSO, H-PSO, and F-PSO in the same situation (column FIT). In almost all of these cases, the superior performance of S-PSO against each of other three algorithms is statistically significant (using pair-wise student t-test). This statistical significance becomes even ~~much~~ more obvious when the complexity of the search space (dimension) ~~gets~~ increases~~d~~.

For PSO initialised with randomized Halton sequence (H-PSO), the results in Tables 1, 2, and 3 are consistent with our preliminary results shown in [18] (even though the experiment and parameter settings are slightly different). The performance of H-PSO is very similar to U-PSO (PSO initialised by the uniform method using a pseudo-random generator). The performance of H-PSO is only ~~got~~ slightly (not statistically significant) better tha~~e~~n U-PSO when the dimension of the search space ~~got~~ increases~~d~~ (except for f_4 in Table 1). In some rare case of high dimensional search spaces, H-PSO was better than the other three algorithms (f_6, n=30 in Table 1) in finding optimal solutions.

For PSO initialised with randomized Faure sequence (F-PSO), the results show that F-PSO was the worst algorithm, finding fewest optimal solutions ~~in~~ and at a slower convergence speed. F-PSO ~~was~~ is only comparable to the other three~~s~~ when the search dimension is small (except for f_6, n=40 in Table 1). In~~t~~ this case, some~~-~~times F-PSO was the best (though the better performances of F-PSO compared to other three are hardly statistically significant). The F-PSO results are not entirely surprising ~~–ed~~ as suggested in the literature [10], the low-discrepancy of the Faure sequence is only guaranteed when the search dimension is small (n<30).

Overall results on the problems ~~tried~~ suggest that randomized low-discrepancy sequences do provide ~~(but not always)~~ a better ~~way~~ initialisation over the traditional and more ~~currently~~ common ~~way of~~ use of~~ing~~ pseudo-random generators, to initialise PSO – but not always. The randomized Sobol sequence seems to be a good candidate based on the results of the experiments in this paper. The randomized Halton sequence might only be useful when the search dimension is high~~, and in contrary~~; by contrast, the randomized Faure sequence s~~d~~eems to be suitable only for low dimension search spaces. The mixed results coming from the use of different randomized low-discrepancy sequences in this paper is understandable. Even though their deterministic versions have been proven to have optimal discrepancies, in practice, their usefulness ~~are~~ is problem dependent [10].

V. Conclusions and Future Work

Overall, the conclusion seems inescapable that, at least for global-best PSO, it is worth replacing uniform sampling of the initial populations with randomized low-discrepancy sequences (randomized Sobol sequence). However, the choice of an arbitrary low-discrepancy sequence initialisation phase does not necessary lead to improvement in the overall performance of PSO (the case of randomized Halton and Faure sequences).

In future, we plan to investigate more thoroughly the reasons behind the success and failure of the three randomized low-discrepancy sequences on the problem tried in this paper, perhaps, through diversity and local fitness landscape studies.

In this paper, we focus only on the initialisation phase in PSO, but it is possible to use randomized low-discrepancy sequences in other phases of PSO, which require random decision making (since they simulate the uniform random distribution). We plan this in the near future. The work reported here could also be extended to other evolutionary and nature-inspired algorithms (such as evolution strategies, differential evolution, continuous ant colony optimization), and we plan to do this at a later date.

Acknowledgment

The authors would like to thank all members of the Natural Computation Group at the Military Technical Academy (MTA), Vietnam, for useful comments and suggestions related to this work. This work is partly funded through a national research grant for fundamental sciences, grant number: 203106

References

[1] E. I. Atanassov, "A New Efficient Algorithm for Generating the Scrambled Sobol' Sequence", in Proceedings of the 5th International Conference on Numerical Methods and Applications, LNCS 2542, Springer-Verlag, 83-90, 2003

[2] E.I. Atanassov, "Effcient CPU-Specific Algorithm for Generating the Generalized Faure Sequences", in *Proceedings of LLSS'2003*, LNCS 2907, Springer-Verlag, 121-127, 2004.

[3] P. Bratley and B. L. Fox, "ALGORITHM 659: Implementing Sobol's Quasirandom Sequence Generator", *ACM Transactions on Mathematical Software*, 29(1), 49-57, 2003.

[4] R. Brits, R., Engelbrecht, A.P, and F. A. Van Den Bergh, "Niching Particle Swarm Optimization", in *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'2002),* 692-696, 2002.

[5] E. Cantu-Paz, "On Random Numbers and the Performance of Genetic Algorithms", in *Proceedings of Genetic and Evolutionary Computation Conference (GECCO) 2002*, 754-761, 2002.

[6] H.M. Chi, P. Beerli, D.W. Evans, and M. Mascagni, "On the Scrambled Sobol Sequence", *in Proceedings of Workshop on Parallel Monte Carlo Algorithms for Diverse Applications in a Distributed Setting*, LNCS 3516, Springer Verlag, 775-782, 2005.

[7] J.M. Daida, D.S. Ampy, M. Ratanasavetavadhana, H. Li, and O.A. Chaudhri, "Challenges with Verification, Repeatability, and Meaningful Comparison in Genetic Programming: Gibson's Magic", in *Proceedings of GECCO 1999*, 1851-1858, 1999.

[8] R.C. Eberhart and J. Kennedy, "A New Optimizer using Particle Swarm Theory", in *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, 39-43, 1995.

[9] A.P. Engelbrechr, *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, 2005.

[10] E.J. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer-Verlag, 1998.

[11] X. Hu, Y. Shi,and R.C. Eberhart, “Recent Advances in Particle Swarm”, in *Proceedings of Congress on Evolutionary Computation (CEC’2004),* 90-97, 2004.

[12] J. Kennedy and R.C. Eberhart, R.C, “Particle Swarm Optimization”, in *Proceedings of the IEEE International Joint Conference on Neural Networks*, 1942-1948, 1995.

[13] S. Kimura, and K. Matsumura, “Genetic Algorithms using Low-Discrepancy Sequences”, in *Proceedings of GECCO 2005,* 1341-1346, 2005.

[14] D.E. Knuth, *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms, Addison-Wesley, 1998.

[15] M.M. Meysenburg and J.A. Foster, “Random Generator Quality and GP Performance”, in *Proceedings of GECCO 1999*, 1121-1126, 1999.

[16] M.M. Meysenburg and J.A. Foster, “Randomness and GA Performance Revisited”, in *Proceedings of GECCO 1999*, 425-432, 1999.

[17] W.J. Morokoff, and R.E. Caflisch, “Quasi-random sequences and their discrepancies”, *SIAM Journal on Scientific Computing*, 15(6): 1251-1279, 1994.

[18] Nguyen Xuan Hoai, Nguyen Quang Uy, and R.I. Mckay, “Initialising PSO with Randomized Low-Discrepancy Sequences: Some Preliminary and Comparative Results”, to appear in *Proceedings of GECCO’2007*, 2007.

[19] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods,* SIAM, 1992.

[20] K.E. Parsopoulos and M.N. Vrahatis, “Particle Swarm Optimization in Noisy and Continuously Changing Environments”, in *Proceedings of International Conference on Artificial Intelligence and Soft Computing,* 289-294, 2002.

[21] K.E. Parsopolous and M.N. Vrahatis, “Initializing the Particle Swarm Optimization using Nonlinear Simplex Method”, in *Advannces in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, WSEAS Press, 216-221, 2002.

[22] W.H. Press, T.A. Teukolsky, W.T. Vetterling., and B.P. Flanenry, *Numerical Recipes in C++: The Art of Scientific Computing*, Cambridge University Press, chapter 7, 2002.

[23] M. Richards and O. Ventura, ”Choosing a Starting Configuration for Particle Swarm Optimization”, in *Proceedings of the Joint Conference on Neural Networks*, 2309-2312, 2004.

[24] M.P. Song and G.C. Gu, “Research on Particle Swarm Optimization: A Reviews”. in *Proceedings of the Third International Conference on Machine Learning and Cybernetics*, 2236-2241, 2004.

[25] X. Wang and F.J. Hickernell, “Randomized Halton Sequences”, *Mathematical and Computer Modelling*, 32: 887-899, 2000.

TABLE 1. NUMBER OF PARTICLES = 50

F	Dim n	U-PSO			H-PSO			S-PSO			F-PSO		
		SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)
f_1	10	100	6412 (527.9)		100	6485 (450)		100	**5705.5** (495.6)		100	5984 (427.8)	
	20	100	18049 (1103.6)		100	18117.5 (1244.1)		100	**15113** (1075.9)		100	19072.5 (1234)	
	30	100	36787.5 (2134.1)		100	35699.5 (2176.1)		100	**29265** (1967.5)		100	38402.5 (2331.5)	
	40	100	62713.5 (3642.3)		100	59881.5 (3652.2)		100	**48094.5** (3053.4)		100	63243.5 (3673.9)	
f_2	10	100	4909 (378.6)		100	4959.5 (412.2)		100	**4368.5** (405.6)		100	4638 (416.3)	
	20	100	16544 (1276.2)		100	16190.5 (1045.8)		100	**13765.5** (1004.9)		100	17053 (1237.8)	
	30	100	35275 (2243.9)		100	34215.5 (2274.1)		100	**28818.5** (1932.1)		100	36741 (2247.9)	
	40	100	61487.5 (3825.0)		100	60284 (3422.6)		100	**49639** (3171.3)		100	63874.5 (3582.3)	
f_3	10	100	8636.5 (567.3)		100	8704.5 (635.9)		100	**7900** (566.3)		100	8201 (573.2)	
	20	100	23662 (1430.5)		100	23433 (1546.9)		100	**20127.5** (1704.8)		4	51687.5 (24786)	19.96 (0.23)
	30	100	46920.5 (3730.7)		100	45517.5 (3511.1)		100	**37897** (2572.7)		0		20.16 (0.19)
	40	99	78499.5 (6218.7)	0.0012 (0)	100	76571.5 (5938.9)		100	**62177** (5156.2)		0		20.29 (0.21)
f_4	10	4	41425 (14854)	**2.76** (1.52)	4	**29037.5** (6500.8)	3.03 (1.57)	**6**	49258.3 (21157)	2.78 (1.30)	3	62833.3 (31403)	3.39 (2.0)
	20	0		15.55 (6.21)	0		15.15 (5.95)	0		**10.32** (3.99)	0		14.2 (5.77)
	30	0		31.88 (10.19)	0		**10.18** (10.43)	0		19.92 (6.81)	0		32.82 (9.54)
	40	0		59.06 (14.75)	0		58.02 (18.56)	0		**29.83** (8.63)	0		57.14 (13.44)
f_5	10	0		1.04 (2.94)	0		0.68 (1.95)	0		**0.07** (0.01)	0		0.41 (2.64)
	20	0		30.64 (24.35)	0		33.61 (29.00)	0		**1.134** (1.13)	0		36.73 (36.82)
	30	0		96.15	0		73.78	0		**4.46**	0		144.67

				(43.78)			(37.24)			(2.88)			(90.07)
	40	0		169.99 (62.13)	0		135.05 (59.63)	0		**10.26** 4.95	0		269.55 (102.7)
f$_6$	10	0		**0.06** (0.03)	0		**0.06** (0.03)	0		0.07 0.03	0		**0.06** 0.03
	20	17	20555.9 (8900.1)	**0.03** 0.02	**19**	26889.5 (18482)	**0.03** 0.02	17	**15150** (1182.7)	0.04 0.03	14	19728.6 (2950.5)	0.04 0.02
	30	27	36700 (7144.8)	**0.02** 0.02	**41**	35592.9 (4747.4)	0.03 (0.02)	33	**29707.6** (8097.5)	**0.02** 0.02	33	39333.3 (10148)	0.03 0.02
	40	49	60421.4 (3652.6)	**0.02** (0.01)	38	57702.6 (3536.1)	0.02 (0.02)	47	**45121.2** (2967.4)	0.02 (0.01)	**59**	61018.6 (3903.1)	**0.02** (0.01)

TABLE 2. NUMBER OF PARTICLES = 100.

F	Dim n	U-PSO			H-PSO			S-PSO			F-PSO		
		SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)
f$_1$	10	100	10984 (675.8)		100	10943 (731.02)		100	9951 (714.85)		100	**9398** (766)	
	20	100	31359 (2012.6)		100	31444 (2063.6)		100	**26041** (1832.9)		100	31546 (2073.7)	
	30	100	63161 (3950.9)		100	62614 (3820.9)		100	**50294** (3156.2)		100	64895 (3818.9)	
	40	17	98282.4 (1190.2)	0.006 (0.007)	23	96934.8 (1998.8)	0.005 (0.005)	**98**	**82623.5** (5345.9)	**0.002** (0.001)	5	96500 (1330.4)	0.012 0.013
f$_2$	10	100	8397 (635.2)		100	8562 (568.7)		100	7539 (587.7)		100	**6880** (561.7)	
	20	100	27966 (1736.5)		100	28029 (1754.5)		100	**23986** (1725.5)		100	28417 (1909.2)	
	30	100	60627 (3671.6)		100	59186 (3619.7)		100	**49883** (3542.7)		100	61855 (3471.8)	
	40	9	98044.4 (1402.8)	0.006 (0.006)	27	96011.1 (3927.5)	0.003 (0.002)	**100**	**87119** (5492)		5	97540 (2326.6)	0.011 (0.022)
f$_3$	10	100	14701 (861.1)		100	15006 (1002.6)		100	13606 (876.8)		100	**13167** (856.6)	
	20	100	40318 (2736.9)		100	40086 (2345.9)		100	**34171** (1896.4)		89	41942.7 (6562.4)	17.93 (5.95)
	30	100	77432 (4110.1)		100	78298 (4119.2)		100	**64767** (4556.1)		2	89250 (2192)	19.9 (1.75)
	40	0		0.023 (0.021)	0		0.020 (0.018)	**24**	**95904.1** (2882.8)	0.003 (0.002)	0		20.33 (0.23)
f$_4$	10	8	55100 (23239)	2.39 (1.28)	11	57109.1 (11688)	**2.03** (1.04)	6	60266.7 (21988)	2.09 (1.04)	**16**	**55075** (22738)	2.04 (0.96)
	20	0		12.85 (4.66)	0		11.93 (4.2)	0		**9.23** (3.67)	0		13.11 (4.13)
	30	0		34.68 (9.38)	0		34.05 (8.9)	0		**20.24** (6.58)	0		35.5 (11.49)
	40	0		76.96 (16.86)	0		75.58 (19.94)	0		**32.6631** (9.47)	0		79 (19.31)
f$_5$	10	0		2.33 (6.04)	0		2.84 (6.1)	0		**0.15** (0.09)	0		0.16 (0.08)
	20	0		40.46 (28.66)	0		37.02 (24.31)	0		**1.94** (1.57)	0		31.77 (23.14)
	30	0		99.06 (41.79)	0		83.63 (35.38)	0		**6.98** (3.62)	0		112.6 (57.72)
	40	0		163.57 (56.84)	0		139.384 (41.63)	0		**14.1164** (5.47)	0		226.09 (90.22)
f$_6$	10	0		0.058 (0.029)	0		0.063 (0.032)	0		**0.055** (0.03)	0		0.063 (0.026)
	20	9	32666.7 (16682)	0.037 (0.03)	4	44400 (23769)	0.031 (0.02)	**15**	**33026.7** (17104)	**0.26** (0.02)	8	38112.5 (11312)	0.04 (0.03)
	30	25	62948 (4175)	0.03 (0.01)	**35**	61260 (3522.1)	**0.02** (0.02)	33	**49500** (3920)	0.03 (0.02)	26	62800 (3779.7)	**0.02** (0.02)
	40	10	96850 (2077.5)	0.02 (0.02)	14	95064.3 (4404.3)	0.01 (0.01)	**45**	79406.7 (4495.6)	0.02 (0.01)	9	97977.8 (1965.2)	0.02 (0.03)

TABLE 3. NUMBER OF PARTICLES = 200.

F	Dim n	U-PSO			H-PSO			S-PSO			F-PSO		
		SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)	SUC	EVAL (STD)	FIT (STD)
f_1	10	100	19102 (1364.9)		100	19298 (1244.8)		100	17276 (1212.7)		100	**16944** (1090.3)	
	20	100	54682 (3271.1)		100	54588 (3654.8)		100	**46146** (2825)		100	50720 (3225.5)	
	30	4	97100 (1669.3)	0.008 (0.008)	9	96600 (2265)	0.007 (0.008)	**99**	**87983.3** (6161.4)	0.001 (0)	6	97966.7 (1997.7)	0.007 (0.006)
	40	0		7.94 (6.43)	0		5.08 (4.38)	0		**0.14** (0.11)	0		8.44 (5.72)
f_2	10	100	14542 (1051.1)		100	14808 (1019.8)		100	13364 (944.68)		100	**12458** (967.9)	
	20	100	48840 (3146.8)		100	48776 (3319.5)		100	**42098** (2811.4)		100	45762 (2607.5)	
	30	20	96430 (2421.2)	0.005 (0.006)	36	96983.3 (2608.3)	0.004 (0.003)	**99**	**86563.6** (5859.9)	**0.001** (0)	16	97500 (2260.4)	0.005 (0.005)
	40	0		10.03 (8.9)	0		6.71 (5.7)	0		**0.27** (0.24)	0		13 (10.1)
f_3	10	100	25974 (1313.7)		100	26154 (1367.7)		100	23814 (1453.2)		100	23678 (1302.6)	
	20	100	68536 3926.6		100	68612 (3674.6)		100	**59398** (3354.2)		100	64250 (3349.2)	
	30	0		0.021 (0.02)	0		0.02 (0.02)	3	97600 (1708.8)	**0.003** (0.002)	0		0.22 (1.99)
	40	0		1.54 (0.69)	0		1.34 (0.67)	0		**0.08** (0.06)	0		19.21 (3.99)
f_4	10	17	80176.5 (12235)	1.69 (0.82)	17	63564.7 (14669)	1.97 (1.05)	15	68080 (16522)	1.77 (0.77)	**27**	**63511.1** (17045)	**1.58** (0.74)
	20	0		14.51 (4.7)	0		14.37 (3.97)	0		**10.18** (3.54)	0		16 (6.23)
	30	0		48.46 (11.9)	0		48.04 (14.38)	0		26.25 (7.8)	0		50.071 (2.37)
	40	0		116 (23.8)	0		108.37 (22.05)	0		**47.20** (14.57)	0		119.06 (27.76)
f_5	10	0		3.23 (4.1)	0		3.35 (5.16)	0		0.31 (0.22)	0		**0.28** (0.13)
	20	0		43.82 (20.24)	0		43.25 (24.04)	0		**3.19** (1.95)	0		23.03 15.27
	30	0		87.78 (33.94)	0		89.57 34.56	0		**10.94** (4.59)	0		76.23 (38.36)
	40	0		153.82 (51.17)	0		149.7 (46.14)	0		**17.72** (5.35)	0		167.32 (61.83)
f_6	10	0		**0.06** (0.03)	0		0.07 (0.03)	0		**0.06** (0.03)	0		**0.06** (0.03)
	20	9	65533.3 (17978)	0.042 (0.03)	**10**	60020 (9245.5)	**0.041** (0.029)	9	54355.8 (17825)	0.043 (0.03)	6	52300 (6079)	**0.041** (0.033)
	30	4	97700 (1113.6)	**0.027** (0.027)	1	94200 (0)	0.029 (0.035)	**30**	**87353.3** (5279.7)	0.03 (0.02)	4	98800 (748.3)	0.028 (0.03)
	40	0		0.83 (0.21)	0		0.73 (0.2)	0		**0.07** (0.06)	0		0.9 (0.16)