Improvement of the Gravitational Search Algorithm by means of Low-Discrepancy Sobol Quasi Random-Number Sequence Based Initialization

O. Tolga ALTINOZ[1], A. Egemen YILMAZ[1], Gerhard-Wilhelm WEBER[2]
[1]*Ankara University, Faculty of Engineering, Electrical-Electronics Engineering Dept., Ankara, Turkey*
[2]*Middle East Technical University, Institute of Applied Mathematics, Ankara,Turkey*
taltinoz@ankara.edu.tr,aeyilmaz@eng.ankara.edu.tr,gweber@metu.edu.tr

***Abstract*—Nature-inspired optimization algorithms can obtain the optima by updating the position of each member in the population. At the beginning of the algorithm, the particles of the population are spread into the search space. The initial distribution of particles corresponds to the beginning points of the search process. Hence, the aim is to alter the position for each particle beginning with this initial position until the optimum solution will be found with respect to the pre-determined conditions like maximum iteration, and specific error value for the fitness function. Therefore, initial positions of the population have a direct effect on both accuracy of the optima and the computational cost. If any member in the population is close enough to the optima, this eases the achievement of the exact solution. On the contrary, individuals grouped far away from the optima might yield pointless efforts. In this study, low-discrepancy quasi-random number sequence is preferred for the localization of the population at the initialization phase. By this way, the population is distributed into the search space in a more uniform manner at the initialization phase. The technique is applied to the Gravitational Search Algorithm and compared via the performance on benchmark function solutions.**

***Index Terms*—evolutionary computation, random number generation, Sobol quasi random number generation, gravitational search algorithm.**

I. INTRODUCTION

The last two decades witnessed the definitions of various methods for solving the optimization problems, as well as suggestions regarding the performance improvement of existing algorithms. The methods defined for improvement can be categorized into three groups, roughly. In the first group, a new operator (for improving diversity in the search phase) is incorporated into the original code [1-3]. The second group of suggestions emphasizes the impact of the algorithm's control parameters [4-6], and the last group aims at modifying the algorithm according to the specific problem requirements [7-9]. By this way, in some cases, an existing disadvantage of the algorithm is removed; in other cases, some powerful aspects of the algorithm are enhanced.

One of the approaches within the first group deals with initialization of the algorithm [10]. Generally, heuristic methods are population-based methods, where each population member (or search agent) is a candidate solution [11]. At the beginning of the algorithm, these candidate solutions must be distributed in the search space; and throughout iterations, the solution is expected to be converging to the optimum (or the optima). Therefore, the initial position of each member is critical for convergence. Thus, the choice of the initial positions in a relatively narrow region causes the solution to be trapped into local optima; whereas, initialization in a wide region might cause poor search capability and slow convergence.

Usually, a random distribution of search members is performed by means of pseudo-random number generators, which are already implemented as libraries in many programming languages and embedded in compilers. In this study, instead of the conventional pseudo-random number generation techniques, random number generation via a "low-discrepancy quasi-random number sequence" will be implemented at the initialization of population, and its performance will be investigated. In other words, the members of the population will be distributed in the search space after selection of random positions via a generated "low-discrepancy quasi-random number sequence". This analysis will be performed for a very recently defined heuristic method, called the Gravitational Search Algorithm.

Gravitational Search Algorithm (GSA) [12], which is a population based nature inspired heuristic algorithm, is proposed by Rashedi et al. in 2009. The motivation of this algorithm is based on performing the search process of the search agents according to the physical relations (particularly, Newton's Law of Gravity) among these agents. Variants of GSA were introduced by the same authors for finding the solution of continuous and discrete (combinatorial) optimization problems [13].

This paper is organized in four sections following the introduction. In Section 2, the basic idea beneath the Sobol quasi-random sequence is explained and demonstrated graphically. Section 3 presents explanation of the Gravitational Search Algorithm and its sub-routines to be modified. Section 4 gives the implementation results and performance comparison of the classical GSA, and the modified version. Finally, Section 5 consists of the concluding remarks and discussions regarding the outcomes and evidences of this study.

Digital Object Identifier 10.4316/AECE.2014.03007

TABLE I. THE 7 UNIMODAL BENCHMARK FUNCTIONS USED IN OUR EXPERIMENT STUDY, WHERE N IS THE DIMENSION OF THE FUNCTION, S IS THE FEASIBLE SEARCH SPACE AND F_{MIN} IS THE MINIMUM VALUE OF THE FUNCTION.

Test Functions	n	S	$f_{\min}$
$f_1(x)=\sum_{i=1}^{n} x_i^2$	30	$[-100,100]^n$	0
$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30	$[-10,10]^n$	0
$f_3(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2$	30	$[-100,100]^n$	0
$f_4(x)=\max\{\lvert x_i\rvert,1\le i\le n\}$	30	$[-100,100]^n$	0
$f_5(x)=\sum_{i=1}^{n-1}(100(x_{i+1}-x_i^2)^2+(x_i-1))^2$	30	$[-30,30]^n$	0
$f_6(x)=\sum_{i=1}^{n}(\lfloor x_i+0.5\rfloor)^2$	30	$[-100,100]^n$	0
$f_7(x)=\sum_{i=1}^{n} ix_i^4+rand$	30	$[-1.28,1.28]^n$	0

II. SOBOL QUASI-RANDOM SEQUENCE

The idea of the quasi-random sequences comes from the problems related to the numerical calculation of the high-dimension integration. A low-discrepancy point set/sequence (quasi-random points/sequence, respectively) is a random point set/sequence that is "less random" compared to a pseudo-random number sequence. On the other hand, such "less randomness" might give the benefit for higher dimension integration approximation, since this low discrepancy sequence samples the high dimension space more uniformly compared to pseudo-random numbers. When applied to an optimization algorithm, this idea might cause an increase at the convergence speed during global optimization; and this constitutes the basic motivation beneath this study.

In general, the actual problem arises from the numerical integration for high-order integrals. There exist two methods for the numerical integration problem: Monte-Carlo and Quasi Monte-Carlo. For numerical integration of order one, there are some classical integration methods like trapezoidal or Simpson's rules. In multidimensional case, this classical integration becomes complicated as the Cartesian products of the single dimensional integration nodes. That way, as the dimension increases, the number of the nodes exponentially increases. To overcome this problem, the Monte-Carlo method, which does not depend on the dimension, was introduced. Actually, Monte-Carlo method is a stochastic method which depends on the randomly selected sampling points in the space. Therefore, in the Monte-Carlo method, some statistical requirements must be satisfied for random numbers in order to obtain high-quality results in integration.

On the other hand, random number generation is performed by means of pseudo-random number generators, which are already implemented as libraries in many programming languages and embedded in compilers. The outputs of such generators do not demonstrate uniformity despite their quality in terms of randomness. This constitutes a drawback during integration since the samples are desired to spread uniformly in the integral space in order to obtain more precise solution. Hence, the idea of applying quasi Monte-Carlo method arises. In current situation, the random numbers in Monte-Carlo method are replaced with the selected deterministic points; this yields smaller error compared to a Monte-Carlo method. The quasi-Monte-Carlo integration is given by:

$$\int_{t_0}^{t_1} f(\tau)d\tau \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n) \tag{1}$$

which contains deterministic points of $x_1,x_2,...,x_N$. Another problem arises from the choice of these deterministic points. There exist two approaches in order to obtain these points: (a) uniform distribution; alternatively, (b) distribution with low-discrepancy (discrepancy is nothing but the measure of the deviation from uniform distribution). In general, the points shall be selected in such a manner that the numerical integration result converges to the precise value; namely:

$$\lim_{N\to\infty}\frac{1}{N}\sum_{n=1}^{N} f(x_n)=\int_{t_0}^{t_1} f(\tau)d\tau \tag{2}$$

Therefore, the points must be selected in such a way that they are uniformly distributed (evenly distributed). Hence, smaller discrepancy results better spacing [14]. That is why a low-discrepancy sequence is needed. In this study, Sobol generator [15] will be used in order to obtain an evenly distributed sequence. The paper [14] presents the implementation of Sobol sequence generator in details.

Previously, in [16], the authors have demonstrated performance improvement in GSA when a relatively-uniform distribution is applied instead of pseudo-random number generator based initialization. And in this study, our aim is now to present a "low-discrepancy sequence initialized GSA" and to compare its performance with the conventional GSA. By this way, initialization of a population-based algorithm under high-dimension problems will be presented. In fact, the idea of the population initialization via quasi-random sequences (instead of pseudo-random number generators) is not brand new. Previously in [17], the authors applied the idea to the genetic algorithm, and obtained promising results. However, they tested this approach for a limited variety of benchmark functions.

TABLE II. THE 6 MULTIMODAL WITH MANY LOCAL MINIMA BENCHMARK FUNCTIONS USED IN OUR EXPERIMENT STUDY, WHERE N IS THE DIMENSION OF THE FUNCTION, S IS THE FEASIBLE SEARCH SPACE AND F_{MIN} IS THE MINIMUM VALUE OF THE FUNCTION.

Test Functions	N	S	f_{min}
$f_8(x)=-\sum_{i=1}^{n}(x_i\sin(\sqrt{\lvert x_i\rvert}))$	30	$[-500,500]^n$	-12569.5
$f_9(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10)^2$	30	$[-5.12,5.12]^n$	0
$f_{10}(x)=-20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{30}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+e$	30	$[-32,32]^n$	0
$f_{11}(x)=\frac{1}{4000}\sum_{i=1}^{n}(x_i-100)^2-\prod_{i=1}^{n}\cos\left(\frac{x_i-100}{\sqrt{i}}\right)+1$ $y_i=1+\frac{1}{4}(x_i+1)$	30	$[-600,600]^n$	0
$f_{12}(x)=\frac{\pi}{30}\left\{10\sin^2(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]+(y_{30}-1)^2\right\}+\sum_{i=1}^{30}u(x_i,10,100,4)$ $y_i=1+\frac{1}{4}(x_i+1)$ $u(xi,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\le x_i\le a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	30	$[-50,50]^n$	0
$f_{13}(x)=0.1\left\{10\sin^2(\pi 3x_1)+\sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+(x_{30}-1)^2\left[1+\sin^2(2\pi x_{30})\right]\right\}+\sum_{i=1}^{30}u(x_i,5,100,4)$ $u(xi,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\le x_i\le a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	30	$[-50,50]^n$	0

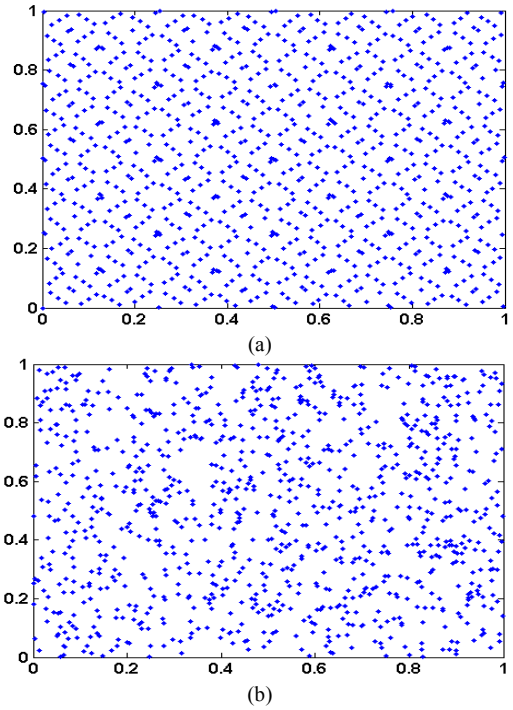

Figure 1. The 2D graphical demonstration of the a) quasi-random (Sobol) sampling b) random sampling on $[0, 1]^2$ space with 1000 samples

Fig. 1 illustrates the distribution of Sobol data set and uniformly distributed pseudo-random data set at two-dimensional space for 1000 data. It clearly demonstrates that the Sobol data are evenly distributed in the space compared to the random data. The difference is clearer for higher data size.

III. MODIFIED GRAVITATIONAL SEARCH ALGORITHM

The heuristic optimization algorithms can be grouped based on their behavior. If an optimization algorithm formulates inspired from natural events, this group of algorithms are called nature-inspired optimization algorithms [18]. As one of the members of this group, Gravitational Search Algorithm (GSA), which is based on the nature interaction, is selected as the optimization algorithm for this study. There are four fundamental interactions existing in the nature: gravitation, electromagnetic force, weak and strong nuclear forces. By taking the "law of gravitation" as the basis of the algorithm, the GSA method was developed.

The law of gravity is the formulation of a force between matters. It gives the relation between impacts of objects, that the force upon on a matter is directly-proportional to the mass of the objects and inversely-proportional to square of the distance between them, as given in (3):

$$F = G\frac{M_1 M_2}{R^2} \tag{3}$$

where G is the gravitation constant, M_1 and M_2 are the masses of the objects, and R is the distance between them. The force upon on an object changes its position by increasing velocity. This relation between force and

TABLE III. THE 10 MULTIMODAL WITH FEW LOCAL MINIMA BENCHMARK FUNCTIONS USED IN OUR EXPERIMENT STUDY, WHERE *N* IS THE DIMENSION OF THE FUNCTION, *S* IS THE FEASIBLE SEARCH SPACE AND F_{MIN} IS THE MINIMUM VALUE OF THE FUNCTION.

Test Functions	*N*	*S*	f_{min}
$f_{14}(x)=\left[\frac{1}{500}+\sum_{i=1}^{35}\frac{1}{j+\sum_{j=1}^{2}(x_i-a_{ij})^6}\right]^{-1}$	2	$[-65,65]^n$	1
$f_{15}(x)=4x_1^2-2{,}1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5,5]^n$	-1.03162
$f_{16}(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10$	2	$[-5.1,5.1]^n$	0.398
$f_{17}(x)=\left[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right]$ $\left[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\right]$	2	$[-2,2]^n$	3
$f_{18}(x)=\sum_{i=1}^{11}\left[a_i-\frac{x_1(b_i^2+b_ix_2)}{b_i^2+b_ix_3+x_4}\right]^2$	4	$[-5,5]^n$	$3e^{-4}$
$f_{19}(x)=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2\right)$	3	$[0,1]^n$	-3.38
$f_{20}(x)=-\sum_{i=1}^{4}c_i\exp\left(-\sum_{j=1}^{6}a_{ij}(x_j-p_{ij})^2\right)$	6	$[0,1]^n$	-3.32
$f_{21}(x)=-\sum_{i=1}^{5}[(X-a_i)^2+c_i]^{-1}$	4	$[0,10]^n$	-10.1532
$f_{22}(x)=-\sum_{i=1}^{7}[(X-a_i)^2+c_i]^{-1}$	4	$[0,10]^n$	-10.4028
$f_{23}(x)=-\sum_{i=1}^{10}[(X-a_i)^2+c_i]^{-1}$	4	$[0,10]^n$	-10.5363

acceleration defined by Newton's second law, which is given in (4) for an object with mass *M*:

$$a=\frac{F}{M} \tag{4}$$

Mass *M* and acceleration *a* are inversely proportional. Hence, when the mass increases, the acceleration decreases, and vice versa. In other words, the smaller mass approaches to the bigger one.

In multi-mass space, each mass influence each other by gravity force. Therefore, general versions of (3) and (4) are defined in (5) and (6), respectively [12], where F_{ij} is the force applied to the *j*th mass by the *i*th mass, and *G* is the gravitational constant:

$$F_{ij}=G\frac{M_{pj}M_{pi}}{R^2} \tag{5}$$

$$a_i=\frac{F_{ij}}{M_i} \tag{6}$$

A very common belief about the evolution of the universe is that our universe is continuously broadening. It is assumed that the universe expands; therefore the distance between any object-pair also increases in time. In an object-pair, if the distance in-between increases, then the mutual force shall decrease. Thus, *G* will decrease by the time as given in (7) [12]. Moreover, (5) becomes more complicated as in (8):

$$G(t)=G(t_0)\left(\frac{t_0}{t}\right)^{\beta} \tag{7}$$

$$F_{ij}=G(t)\frac{M_{pj}M_{pi}}{R^2} \tag{8}$$

where $\beta<1$. At this point, it should be noted that, even though *G* is referred to as the "gravitational constant", it will no more be a constant as seen in (8) under these assumptions. Nevertheless, throughout the paper, we will continue referring to *G* as the "gravitational constant" in the conventional manner. The physical relations among force, velocity and mass cause the gravity alternation with time. In the meantime, the mass and acceleration of each object, calculated from the law of motion and the law of velocity, undergo a change. Hence, that is the motivation source of the GSA algorithm, which is proposed by using these equations and relations.

The fundamental nature of the GSA algorithm is composed of four steps [19]:

1. initialization of the population of agents (objects);
2. fitness evaluation for each agent;
3. updates and calculations by using physical laws;
4. repeating the 2nd and the 3rd steps until the termination condition is met.

From the steps given above, only step 1 is modified by changing the pseudo-random number generator.

Step 1: *Initialization of the Population of Agents (Objects)*

In the conventional GSA, the positions of *N* number of agents (objects) for *n* dimensional search space are randomly initialized with a pseudo-random number generator between the boundaries of the search space at the

TABLE IV. THE 7 SHIFTED/ROTATED BENCHMARK FUNCTIONS (30 DIMENSION) USED IN OUR EXPERIMENTAL STUDY, WHERE O IS THE SHIFTED VALUE FOR THE OPTIMUM, MO IS THE ORTHOGONAL MATRIX FOR ROTATION, S IS THE FEASIBLE SEARCH SPACE AND FMIN IS THE MINIMUM VALUE OF THE FUNCTION.

Test Functions + f_{bias}	z	S	$f_{\min}$
$f_{24}(x)=\sum_{i=1}^{n}(A_i-B_i(x))^2$	$z=x-o$	$[-\pi,\pi]^n$	-460
$f_{25}(x)=\sum_{i=1}^{n}\left((10^6)^{\frac{i-1}{n-1}}z_i^{\,2}\right)^2$	$z=(x-o)M_o$	$[-100,100]^n$	-450
$f_{26}(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}z_j\right)^2(1+0{,}4\lvert N(0,1)\rvert)$	$z=x-o$	$[-100,100]^n$	-250
$f_{27}(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}z_j\right)^2$	$z=x-o$	$[-100,100]^n$	-450
$f_{28}(x)=\sum_{i=1}^{n-1}(100(z_{i+1}-x_i^{\,2})^2+(z_i-1))^2$	$z=x-o+1$	$[-100,100]^n$	390
$f_{29}(x)=\frac{1}{4000}\sum_{i=1}^{n}(z_i-100)^2-\prod_{i=1}^{n}\cos\left(\frac{z_i-100}{\sqrt{i}}\right)+1$ $y_i=1+\frac{1}{4}(x_i+1)$	$z=(y-o)M_o$	$[0,600]^n$	-180
$f_{30}(x)=\sum_{i=1}^{n}(z_i^{\,2}-10\cos(2\pi z_i)+10)^2$	$z=(x-o)M_o$	$[-5,5]^n$	-330

initial iteration t=0, where t={0,1,2,...,$t_{\max}$}. However, in this study, instead of a pseudo-random number generator, Sobol data set are preferred. For both methods, the initial position (X) and initial velocity (V) of the ith agent are defined as follows:

$$X_i(t)=(x^1{}_i(t),x^2{}_i(t),...,x^n{}_i(t)) \tag{9}$$

$$V_i(t)=(v^1{}_i(t),v^2{}_i(t),...,v^n{}_i(t)) \tag{10}$$

Step 2: *Fitness Evaluation for Each Agent*

The fitness function ($fit_i(t)$) is evaluated for each agent j Table I and II present lists of fitness functions which are evaluated in this paper for minimization problem; $\text{argmin}_x(f(X_i))$, where i=1,...,N at each iteration t and stored in memory. After evaluations, the best and worst fitness values are obtained from (11) and (12), respectively:

$$best(t)=\min_{j=1,...,N}(fit_j(t)) \tag{11}$$

$$worst(t)=\max_{j=1,...,N}(fit_j(t)) \tag{12}$$

In the GSA, the fitness values of each agent, the best and worst fitness values are evaluated to assign mass for each agent. The large masses mean better agents with relatively small fitness values, the small masses correspond to worst agents with relatively large fitness values.

Step 3: *Updates and Calculations*

Steps 1 and 2 are very common for similar nature-inspired optimization algorithms (i.e., particle swarm optimization). However, both methods become different at this step, where position and/or other particle properties are altered in this stage. In this step, gravitational "constant", applied force on each particle, distance between masses, new velocity and new position of all particles are computed.

The gravitational constant varies along the iterations. The gravitational constant $G(t)$, velocity V and position X are updated; the mass M and acceleration of the agents are computed. The gravitational constant G is computed from (13):

$$G(t)=G_0e^{-\alpha t/t_{\max}} \tag{13}$$

where t is the time variable, corresponds to current iteration; $t_{\max}$ is the maximum iteration, G_0 is the initial gravitational constant (set to 100), and α is the algorithm control variable (set to 20) [6]. The mass of each agent is constructed according to its fitness. In (14) the fitness value of the ith agent is normalized between best and worst fitness values of all population at the current iteration. Thus, the mass of ith agent is calculated from $fit_j(t)$ value of each agent. Therefore the fitness values of population are normalized in [0,1], where 0 means that the agent has the worst fitness value, similarly, 1 means that is has the best fitness value among the whole population. By the definition of (15), the sum of all normalized values become unity (The sum of all masses/universe becomes unity by this formulation).

$$m_i(t)=\frac{fit_i(t)-worst(t)}{best(t)-worst(t)} \tag{14}$$

$$M_i(t)=\frac{m_i(t)}{\sum_{j=1}^{N}m_j(t)} \tag{15}$$

where M_i is the normalized mass of the ith agent. The acceleration of the ith agent is computed directly by using (6), where the acceleration depends on the force applied to an agent. Thus, primarily, the force F_{ij} applied by the ith mass to the jth one at the dimension n is calculated via (16):

TABLE V. THE 30 BENCHMARK FUNCTIONS USED IN COMPARISON BETWEEN CONVENTIONAL GSA AND SOBOL-GSA ALGORITHMS BASED ON MEAN BEST FITNESS (*MF*), STANDARD DEVIATION (*SD*), BEST (*B*), AND WORST (*W*) OPTIMUM VALUES FOR 50 INDEPENDENT RUNS.

Func.	GSA				Sobol-GSA			
	MF	*SD*	*B*	*W*	*MF*	*SD*	*B*	*W*
f_1	177.7004	124.8005	26.5186	556.0885	0	0	0	0
f_2	0.14647	0.33494	4.1086e-8	1.8944	0	0	0	0
f_3	1101.3305	517.8541	488.9794	2739.5361	0	0	0	0
f_4	8.7763	1.6418	6.027	12.6601	0	0	0	0
f_5	355.7593	401.1153	32.284	2455.806	28.8836	0.39142	26.8602	29
f_6	369.7	190.061	69	826	0	0	0	0
f_7	0.051294	0.029094	0.0095005	0.1816	0.055473	0.028806	0.006339	0.13754
f_8	-2728.6023	383.5772	-3565.1249	-1805.754	-7506.903	260.027	-8133.8942	-7025.3361
f_9	18.5359	4.8938	10.9445	29.8488	0	0	0	0
f_{10}	0.027435	0.089776	5.3065e-9	0.35711	8.881e-16	0	8.881e-16	8.881e-16
f_{11}	168.4556	28.0281	81.1541	207.6757	0	0	0	0
f_{12}	2.2109	0.99297	0.55883	4.6405	0.061416	0.069812	2.8808e-9	0.29786
f_{13}	22.5079	10.2072	2.4648	45.5253	0.93685	0.76739	0.013017	3
f_{14}	6.2841	3.36196	1.0057	14.2686	4.2772	1.4968	1.1099	5.3525
f_{15}	-1.0316	4.3145e-16	-1.0316	-1.0316	-1.0316	4.2082e-16	-1.0316	-1.0316
f_{16}	0.39789	3.3645e-16	0.39789	0.39789	0.39789	3.3645e-16	0.39789	0.39789
f_{17}	3	3.5854e-15	3	3	3	4.7404e-15	3	3
f_{18}	6.68e-3	4e-3	1.48e-3	20.3e-3	5.86e-3	3.39e-3	1.16e-3	17.4e-3
f_{19}	-3.8609	3.05e-3	-3.8628	-3.8464	-3.861	2.72e-3	-3.8628	-3.8482
f_{20}	-3.3159	3.03e-3	-3.322	-3.1608	-3.3189	2.16e-3	-3.322	-3.169
f_{21}	-6.9569	3.6808	-10.1532	-2.6305	-6.9685	6.6654	-10.1532	-2.6829
f_{22}	-10.1358	1.322	-10.4029	-3.7243	-10.1431	1.3062	-10.4029	-2.7659
f_{23}	-10.2208	1.5625	-10.5364	-2.4217	-10.0778	1.841	-10.5364	-2.4217
f_{24}	9.93e4	5.03e4	2.75e4	2.34e5	10.2e4	4.29e4	2.1e4	2.15e5
f_{25}	1.34e9	2.61e8	9.21e8	1.93e9	1.32e9	1e8	10.4e8	1.52e9
f_{26}	1.22e5	3.66e4	6.15e4	2.1e5	1.5e5	3.08e4	9.4e4	2.13e5
f_{27}	8.09e4	2.04e4	4.93e4	1.41e5	1.25e5	2.33e4	7.85e4	1.64e5
f_{28}	189e10	3.1e9	1.33e10	2.53e10	2.3e10	1.13e9	2.12e10	2.56e10
f_{29}	12.1e4	0.6e3	1.1e4	1.3e4	4.6e3	0	4.6e3	4.6e3
f_{30}	-257.4	16.4	-288.2	-223.5	-261.7	13.19	-287.6	-224.5

$$F^n{}_{ij} = G(t)\frac{m^2{}_i(t)}{\left\|X_i(t), X_j(t)\right\|_2 + \varepsilon}(x^n{}_j(t) - x^n{}_i(t)) \quad (16)$$

where ε>0 is a small constant. In GSA, the total force applied on agent *i* is calculated as randomly weighted sum of the fitness values of all agents (in (17)) where *rand* is a random variable uniformly distributed in [0,1] and produced by a pseudo-random number generator:

$$F_i^n = \sum_{j=1}^{N}(rand_j F^n{}_{ij}(t)) \quad (17)$$

In summary, the total sum of the weighted force applied on a particle with a random number is found. Finally, the acceleration of the agent *i* at dimension *n* is calculated from (6) and defined as (18):

$$a^n{}_i(t) = \frac{F^n{}_i(t)}{M_i(t)} \quad (18)$$

At the beginning of the new iteration, the positions and velocities are calculated by using (19) and (20):

$$v^n{}_i(t+1) = rand_i v^n{}_i(t) + a^n{}_i(t)\delta t \quad (19)$$

$$x^n{}_i(t+1) = x^n{}_i(t) + v^n{}_i(t)\delta t \quad (20)$$

Step 4: *Repeat*

If the end criterion (such as reaching the maximum number of iterations) is met then the solution is picked as the position of the best agent, and the program is terminated; else Step 2 is re-executed, and the process is repeated by taking the last population as the initial population of the new iteration by incrementing the iteration index.

IV. IMPLEMENTATION AND RESULTS

In this study, the initialization phase GSA is altered by applying the "quasi-random number generator" instead of the pseudo-random number generator for distribution of the search agents; then, the modified GSA will be applied to 30 benchmark functions, and results will be compared to previously obtained conventional GSA outputs. Tables I-III present the benchmark functions in three different categories: unimodal, multimodal with many local minima, and multimodal with few local minima, and Table IV gives the rotated/shifted benchmark problems (detailed information related to benchmark functions can be obtained in [20]). In this study, instead of conventional benchmark problems, which has the global optimum point generally located in the center of the search space, or at zero, the shifted and rotated problems are preferred such that the shape and optimum of conventional problems is altered since almost all cases the real-word problems have the solution far from zero and center of search space.

Two implementations are executed in this study: the conventional Gravitational Search Algorithm (GSA); and

the GSA with quasi-random number generator, which is named as Sobol-GSA. The parameters of these two methods are set to be equal: the number of particles is 50 and the number of iterations is 250. For a fair performance comparison, 50 independent runs are executed, and the average, best and worst values as well as the standard deviation are taken into consideration. Table V presents the relevant results.

The results presented in Table V are categorized as the mean of the best fitness values (MF) of 50 independent Monte-Carlo runs, standard deviation of all executions (SD) and best/worst fitness values (B and W, respectively) among the results. The results indicate that Sobol-GSA is superior to GSA.

For unimodal benchmark functions f_1-f_4 and f_6, Sobol-GSA obtained global optimum at every independent Monte-Carlo run. For other unimodal benchmark functions except mean best fitness value of f_7, Sobol-GSA outperforms GSA.

For multimodal benchmark functions with many local optimum f_9 and f_{11}, Sobol-GSA obtains a global optimum at every independent run. Furthermore, it outperforms GSA for other benchmark functions. When the results demonstrated in Table V are investigated, it is observed that, GSA and Sobol-GSA present the same performance for benchmark functions f_{15}, f_{16} and almost for f_{17}. However, for f_{14} Sobol-GSA performs better. The results for the rotated/shifted benchmark problems demonstrate that the proposed initialization scheme might demonstrate worse performance for most cases. On the other hand, the obtained results are quite close to each other.

The results for the benchmark functions f_1, f_2, f_3, f_4, f_6, f_9, f_{11} and f_{17} in Table V indicate that at every independent run of the Sobol-GSA, the algorithm can detect the global optimum, possibly at the beginning or early phase of the iterations. This reason behind these phenomena is explained as regards the comparison of pseudo random and Sobol distributed particles on problem contours. For this reason the benchmark problems f_1, f_8 and f_9 are selected as test beds because of their various contour shapes. Figs. 2-4 show the contour plots of the case problems.

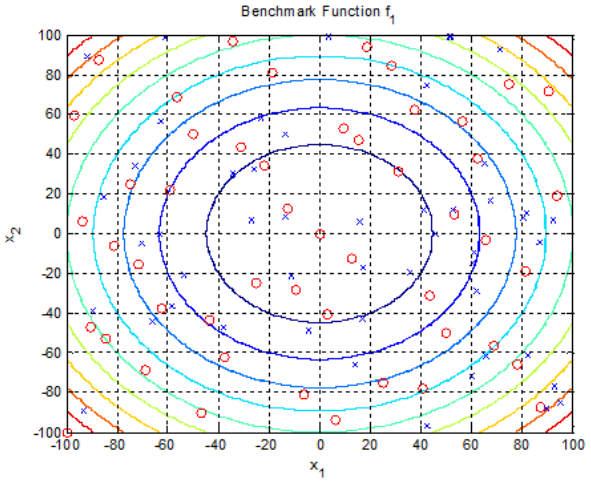

Figure 2. The contour description of the benchmark problem f_1 and scatter particles both uniform pseudo-randomly (‘X’) and Sobol quasi-randomly (‘O’)

In Fig. 2, the contour plot and position of particles demonstrate the distribution of the Sobol numbers, some of them are located at (near to) the global optimum (0,0). Therefore, for f_1 (or similar functions: f_2, f_3, f_4, f_6) at the initialization (or at the first few iterations), the algorithm reaches the global optimum.

The benchmark functions f_8 and f_9 are the examples which have many local optimums. As the number of local optimum increases, the number of the circles and nested circles also increases. This turn out that the size of the smaller circle on the contour plot (which is the area of the local/global optimum) lessens, which yield that finding the global optimum becomes harder for proposed initialization. However, even for these cases, the proposed algorithm shows better performance than common initialization.

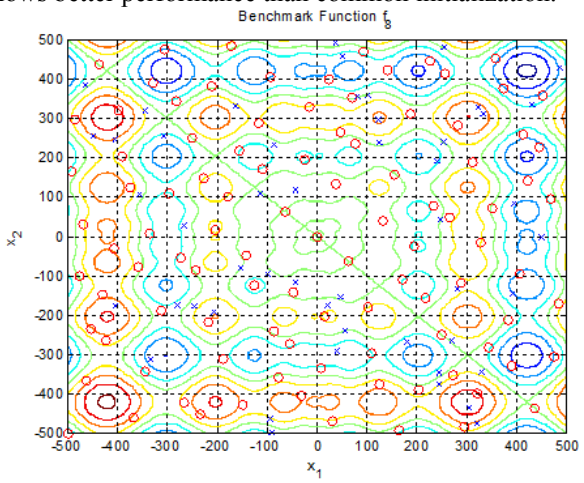

Figure 3. The contour description of the benchmark problem f_8 and scatter particles both uniform pseudo-randomly (‘X’) and Sobol quasi-randomly (‘O’)

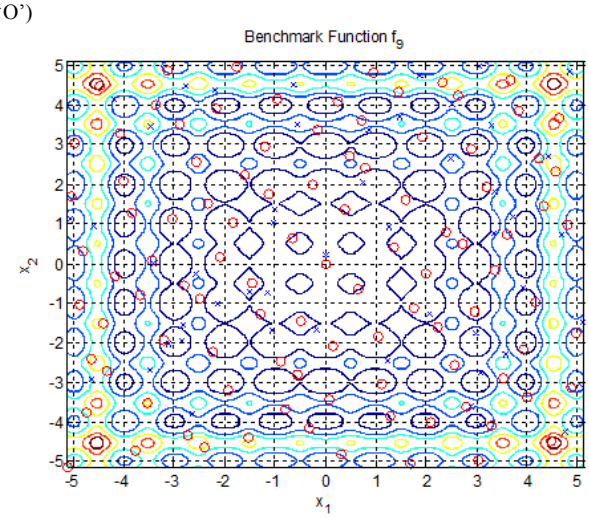

Figure 4. The contour description of the benchmark problem f_9 and scatter particles both uniform pseudo-randomly (‘X’) and Sobol quasi-randomly (‘O’)

V. Conclusion

This paper presents improved version of the GSA by changing the initialization phase by substituting the pseudo-random number generator with a quasi-random number generator based on the Sobol data set. Simulation results were obtained for the conventional GSA and the so-called Sobol-GSA for 30 benchmark functions. From these results, it can be clearly seen that the Sobol set improves the performance of the optimization algorithm for unimodal benchmark functions and multimodal functions with many local optima. However, for multimodal functions with a few

local optimum Sobol-GSA and GSA present almost the same performance. From the results obtained in this study and the previous studies of the authors, it can be concluded that, improvement in distribution of the particles at the initialization phase has a considerable positive performance impact for the solution of unimodal functions and multimodal functions with many local optima.

The random number generation method used throughout this study does not rely on any in-complier embedded pseudo-random number generator. This is a significant advantage especially on the currently evolving parallel and distributed architectures, such as GPGPU (General Purpose Graphics Processing Unit) or grid structures, in which the message traffic due to the transfer of the generated random number constitutes a bottleneck. Via the quasi-random number generators, the random numbers can be directly generated on the relevant core (or the processing unit), without any need of data transfer among the cores. Hence, the proposed method might be applicable to research areas, such as cryptology, financial mathematics, stochastic problems on signal processing and power electronics, in which accurate and fast random number generation techniques are desired.

On the other hand, for functions with a few local optimum and shifted/rotated functions, the performance does not improve dramatically. As a future study, the authors will focus on this issue.

ACKNOWLEDGMENTS

This study is made possible by a grant from TUBITAK (with Grant Nr. 112E168). The authors would like to express their gratitude to TUBITAK for their support.

REFERENCES

[1] Y.W. Leung, Y. Wang, Y.W. Leung, "An orthogonal genetic algorithm with quantization for global numerical optimization," IEEE Transactions on Evolutionary Computation, Vol. 5, No. 1, pp. 41-53, 2001.

[2] O.T. Altinoz, A.E. Yilmaz, G.W. Weber, "Application of chaos embedded PSO for PID tuning," International Journal of Computers, Communications and Control, Vol. 7, No. 2, pp. 204-218, 2012.

[3] E. Masahian, D. Sedighizadeh, "Multiobjective particle swarm optimization and NPSO-based algorithms for robot path planning," Advances in Electrical and Computer Engineering, Vol. 10, No. 4, pp. 69-76, 2010.

[4] A. Ratnaweera, S.K. Halgamuge, H.C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," IEEE Transactions on Evolutionary Computation, Vol. 8, No. 3, pp. 240-255, 2004.

[5] A. Morales-Reyes, A.T. Erdogan, "A structure based coarse fine approach for diversity tuning in cellular GAs," Advances in Electrical and Computer Engineering, Vol. 12, No. 3, pp. 39-46, 2012

[6] G. Mortinovic, D. Bojer, "Elitist ant system with 2-opt local search for the traveling salesman problem," Advances in Electrical and Computer Engineering, Vol. 12, No. 1, pp. 25-32, 2012.

[7] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," IEEE Transactions on Evolutionary Computation, Vol. 10, No. 3, pp. 281-295, 2006.

[8] O.T. Altinoz, A.E. Yilmaz, "Particle swarm optimization with parameter dependency walls and its sample application to the microstrip-like interconnect line design," AEÜ-International Journal of Electronics and Communications, Vol. 66, No. 2, pp. 107-114, 2012.

[9] O. Brudaru, D. Popovich, C. Copecanu, "Cellular genetic algorithm with communicating grids for assembly line balancing problems," Advances in Electrical and Computer Engineering, Vol. 10, No. 2, pp. 87-93, 2010.

[10] B. Liu, L. Wang, Y.H. Jin, "An effective PSO-based memetic algorithm for flow shop scheduling," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol. 37, No. 1, pp. 18-27, 2007.

[11] J. Kennedy, Y. Shi, R. Eberhart, Swarm Intelligence. San Diego, CA, USA: Academic Press, 2001.

[12] E. Rashedi, H. Nezamabadi, S. Saryazdi, "GSA: A gravitational search algorithm," Information Sciences, Vol. 179, No. 13, pp. 2232-2248, 2009.

[13] E. Rashedi, H. Nezamabadi, S. Saryazdi, "BGSA: Binary gravitational search algorithm," Natural Computing, Vol. 9, No. 3, pp. 727-745, 2010.

[14] P. Bradley, B.L. Fox, "Algorithm 659: Implementing Sobol's quasirandom sequence generator," ACM Transactions on Mathematical Software, Vol. 14, No. 1, pp. 88-100, 1988.

[15] I.M. Sobol, "Distribution of points in a cube and approximate evaluation of integrals," Zhurnal Vychislitelnoi Matematik i Matematicheskoi Fiziki (USSR Computational Mathematics and Mathematical Physics), Vol. 7, No. 4, pp. 784-802, 1967.

[16] O.T.Altinoz, A.E. Yilmaz, G.W. Weber, "Orthogonal array based performance improvement in the gravitational search algorithm," Turkish Journal of Electrical Engineering and Computer Sciences, Vol. 21, No. 1, pp. 174-185, 2013.

[17] H. Maaranen, K. Miettinen, M.M. Makela, "Quasi-random initial population for genetic algorithms," Computers & Mathematics with Applications, Vol. 47, No. 12, pp. 1885-1895, 2004.

[18] S.A. Kazarlis, A.G. Bakirtzis, V.A. Petridis, "A genetic algorithm solution to the unit commitment problem," IEEE Transactions on Power Systems, Vol. 11, No. 1, pp. 83-92, 1996.

[19] E. Rashedi, H. Nezamabadi, S. Saryazdi, "Filter modeling using gravitational search algorithm," Engineering Applications of Artificial Intelligence, Vol. 24, No. 1, pp. 117-122, 2011.

[20] P.N. Suganthan, N. Hansen, J.J. Liang, K.Deb, Y.P. Chen, A. Auger, S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real parameter optimization," 2005 IEEE Congress on Evolutionary Computation (CEC 2005), pp. 1-5, 2005.