Available online at www.alphanumericjournal.com

alphanumeric journal
The Journal of Operations Research, Statistics, Econometrics and Management Information Systems

Volume 8, Issue 1, 2020

Received: June 12, 2019
Accepted: June 05, 2020
Published Online: June 30, 2020

AJ ID: 2020.08.01.OR.02
DOI: 10.17093/alphanumeric.576919
Research Article

Machine Coded Compact Genetic Algorithms For Real Parameter Optimization Problems

Mehmet Hakan Satman, Ph.D. *

Prof., Department of Econometrics, Faculty of Economics, İstanbul University, İstanbul, Turkey, mhsatman@istanbul.edu.tr

Emre Akadal, Ph.D.

Res. Assist., Department of Informatics, İstanbul University, İstanbul, Turkey, emre.akadal@istanbul.edu.tr

* İstanbul Üniversitesi İktisat Fakültesi Ekonometri Bölümü Rektörlük Merkez Bina Beyazıt, Fatih İstanbul Türkiye

ABSTRACT

In this paper, we extend the Compact Genetic Algorithm (CGA) for real-valued optimization problems by dividing the total search process into three stages. In the first stage, an initial vector of probabilities is generated. The initial vector contains the probabilities of bits having 1 depending on the bit locations as defined in the IEEE-754 standard. In the second stage, a CGA search is applied on the objective function using the same encoding scheme. In the last stage, a local search is applied using the result obtained by the previous stage as the starting point. A simulation study is performed on a set of well-known test functions to measure the performance differences. Simulation results show that the improvement in search capabilities is significant for many test functions in many dimensions and different levels of difficulty.

Keywords: Optimization, Genetic Algorithms, Evolutionary Optimization, Simulations

1. Introduction

Genetic Algorithms (*GA*s) are search and optimization techniques that mimic the natural selection and principals of genetics (Holand, 1975; Goldberg and Holland, 1988; Sastry, 2014). In *GA*s, a population of random solutions are generated and assigned to fitness values. A fitness value is a measure of the quality of a candidate solution. Well known genetic operators such as crossover and mutation are applied on the selected candidate solutions which have higher fitness values to generate new population members called offspring. After many steps, it is expected that the generated population will have higher average fitness than the one generated in former iterations (Goldberg, 1989).

Estimation of Distribution Algorithms (*EDA*s) form another family of *GA*s in which a vector of probabilities are used to generate candidate solutions by sampling rather than a population of candidate solutions (Pelikan et al., 2015). *PBIL* (Population-based incremental learning) is an earlier member of *EDA*s that consists on creating a population of candidate solutions by sampling and updating the vector of probabilities using some best solutions (Baluja, 1994). A vector of probabilities is initially created as $[0.5\ 0.5 \dots 0.5]$. ith element of the vector represents the probability of $P(b_i = 1)$ where b_i is the ith bit of the candidate solution, $i = 1,2,\dots,l$, and l is the number of elements. The best nv solutions are selected from the population of size n to update the vector of probabilities. The aim of the update process is to increase or decrease the probabilities towards to best solutions for generating better solutions in following iterations. After many steps, it is expected that the elements of the probability vector approach to either zero or one. The final solution is a bit string which is considered as the optimum.

Compact Genetic Algorithms (*CGA*s) are the other branch of the *EDA* family (Harik et al., 1999). *CGA*s are compact as they are not based on a population and require less computer memory to run. This property of *CGA*s makes the hardware implementation possible in devices with low resources (Aporntewan and Chongstitvatana, 2001). In each step of the algorithm, two candidate solutions are sampled using the vector of probabilities. Depending the fitness values, the best candidate solution is labeled as the *winner*. If ith gene of the winner is 1, then the ith element of the probability vector is moved towards to 1 with the amount of $\frac{1}{\text{popsize}}$ where $i = 1,2,\dots,n$, n is the chromosome length, and popsize is the population size. If the ith gene of the *winner* is 0, then the amount of mutation is negative, that is, the ith element of the probability vector is moved downwards to 0. These operations are repeated until all of the elements of the probability vector are either 0s or 1s. If the *popsize* parameter is large then the amount of mutation is low, that is, more computation time is needed to get a fully converged probability vector. When the *popsize* is small, then the changing steps are large, convergence rate is high but the result is generally a local optimum because the search space is not well explored. Since the parameters of the crossover probability, mutation probability, population size, number of generations, and crossover and mutation types are not needed, *CGA*s are parameterless. The *popsize* parameter is only about the mutation of probability elements and it is not really defines the number of candidate solution as in *GA*s.

Classical *GA*s and *CGA*s represent the search space using bits. Addition to this, both *GA*s and *CGA*s are extended to use other types of encoding systems such as integer encoding, floating-point or real-valued encoding, permutation encoding, machine-coding, etc. *PBIL* and *CGA* are mainly developed for the binary encoding of variables.

Since it is possible to encode real values as bits, these algorithms can also be applied on the real valued optimization problems. Besides this, some new sampling schemes are based on sampling values using some probability distributions and mutating the distribution parameters during iterations (Sebag and Ducoulombier, 1998; Mininno et al., 2008).

In this paper, we devise a new *CGA* based algorithm for the real valued optimization problems. The encoding of variables is based on binary encoding but the *IEEE-754* transformation is used to separate the *sign*, the *exponent*, and the *mantissa* parts of a real value as stored in computer memory. The algorithm starts with an adjusted probability vector. The adjusted vector is the vector of probabilities in which the corresponding elements represent the probability of bits having value of 1 depending on the locations of bits in the *IEEE-754* standard. After obtaining the adjusted vector, the usual *CGA* search is performed. Finally, a local search is applied to obtain more precise solutions.

In Section 2, we present the algorithm in great detail. In Section 3, an example is given to demonstrate results of the each phase applied on a well known test function. In Section 4, we perform a simulation study to measure the performance differences between the original and the extended algorithms. Finally, in Section 5, we conclude.

2. The Algorithm

The extended algorithm is mainly based on three steps. In first step, an initial vector of probabilities is generated using the *IEEE-754* encoded bits of variables. The initial vector of probabilities does not necessarily have 0.5 in each elements. In the second step, a *CGA* search is performed using the same encoding scheme of real values. In the last step, a local search is applied to obtain more precise solutions. These steps are defined in Section 2.1, Section 2.2, and Section 2.3.

2.1. Encoding of variables

Digital computers store and represent the data using bits. Since bits are numbers in base 2, it is straightforward to express integer numbers by combining many bits. Representing rational numbers is also possible using a finite number of bits. However, representation of real or irrational numbers requires a discretization process (Goldberg, 1991). Emphasizes that success of a *GA* search is related to the building blocks represented by bits as proved in *Schemata* theorem. Since there is not a distinction between *phenotype* and *genotype* of variables, real valued *GA*s are *blocked* in later iterations.

IEEE-754 is a standard for encoding and decoding real numbers using fixed number of bits in computer memory (IEEE, 2008). In this standard, bits of a 32-bit floating number is divided into three parts. The first part is 1 bit length and defines the *sign* of the number. The following 8 bits form the *exponent* part and the remaining 23 bits form the *mantissa*. Finally a 32 bit floating-point number is defined as

$$(-1)^{sign} \times 2^{exponent} \times mantissa$$

where $sign$ is zero if the number is positive. Table 1 shows an example of how the bit representation is changed when a single digit is changed. Since there are 2^{32} possible representations, the first bit divides the total number of possibilities by 2. It is also shown that the *exponent* part remains same when a small change is occurred in the number in some cases. In contrast, the numbers 12345.6789 and 02345.6789 have several differences in both *exponent* and *mantissa* parts even they differ in a single digit. Consequently, numbers sampled in a predefined range have some patterns in both *sign*, *exponent*, and *mantissa* parts.

Using machine based transformations as the encoding scheme is not new in evolutionary optimization context. Budin et al., (2010) used the 64-bits version of the *IEEE-754* standard in *GA* search. It is shown that the machine based encoding scheme outperforms the classical binary encoding. Ojha et al. (2012) trained a neural network using a *GA* search with the variables encoded by 32-bits *IEEE-754* standard. Umbarkar et al. (2015) developed a software based *GA* that uses the *IEEE-754* standard for the encoding scheme. Similarly, Satman (2013) suggested using the byte representation of double precision real values and showed that the byte based encoding outperforms the real-valued encoding scheme in many cases (Satman and Akadal, 2017)

Number	Sign	Exponent	Mantissa
-12345.6789	1	10001100	10000001110011010110111
12345.6789	0	10001100	10000001110011010110111
1234**4**.6789	0	10001100	10000001110001010110111
123**3**5.6789	0	10001100	10000001011111010110111
12**2**45.6789	0	10001100	01111110101011010110111
1**1**345.6789	0	10001100	01100010100011010110111
02345.6789	0	10001010	00100101001101011011101
12345.678**8**	0	10001100	10000001110011010110111
12345.67**7**9	0	10001100	10000001110011010110110
12345.6**6**89	0	10001100	10000001110011010101101
12345.**5**789	0	10001100	10000001110011001010001
10000.0000	0	10001100	00111000100000000000000

Table 1. IEEE-754 representation of some 32-bit floating-point numbers

Suppose the single variable function $y = f(x)$ has an extremum at $x = x_0$ where $-50 \leq x \leq 50$. Let S_i is the *IEEE-754* encoded bit string of the ith floating-point number in the defined range, $i = 1,2,\ldots,m$, and m is the total number of floating-point numbers. Then the probability of the sign bit having the value of 1 is 50% because half of the values lies above the zero. The interesting part is the *exponent* as the values in the predefined range possibly have 1s in the first bit whereas the second bit is generally zero. Note that in the *mantissa* part, most of the bits can be either 0s or 1s with the probability of 50% except the first one since the probability of first bit having 1 is 36%. The probability vector of the *exponent* part is given in Table 2.

1	2	3	4	5	6	7	8
0.96	0.04	0.04	0.04	0.04	0.04	0.51	0.43

Table 2. $P(b_{ij} = 1)$ in exponent parts of $-50 \leq x \leq 50$

In *CGA*s, initial elements of the probability vector are both set to 0.5. As it is mentioned before, this assumption is not really needed and the some parts of the search space is not needed to be explored. Addition to this, some bits can be either 0 or 1, however the probabilities of having 0 or 1 are not equal in some cases. In the first part of the devised algorithm, the adjusted probability vector is generated before the genetic search in order to prevent moving around the unnecessary parts of the search space. When the range of the variable x is defined as $-\infty < x < \infty$ then all of the elements of the probability vector are 0.5. In this special case, the proposed algorithm does not start with an *initial probability generating* process.

2.2. Generating initial probability vector

As mentioned in Section 2.1, the proposed algorithm is based on the *IEEE-754* transformation on the real values of 32 bits. Since any bits of an real value can be 1 with probability of $\frac{1}{2}$ in a range of $-\infty < x < \infty$, probabilities for some bits can be different in a more narrowed range. For instance, $P(b_1 = 0)$ is always zero for the range of $10 \leq x \leq 100$ whereas $P(b_1 = 1)$ is 1 for any range with both elements are negative, where b_i is the ith bit of the *IEEE-754* representation.

The proposed algorithm estimates the probability vector by generating the empirical probabilities of having $P(b_i = 1)$. Algorithm 2.2 shows the whole process for a single variable using a *pseudo-code*. The process can be repeated for the other variables in the multivariate case. The algorithm generates a probability vector of size 32 for a single real variable. If the bounds of the variable is defined as $[minval, maxval]$, *B* samples are sampled using a Uniform distribution with parameters *minval* and *maxval*. The *encode()* function gets a real number as argument and returns the *IEEE-754* representation. In the iteration i, the bit vector is appended at the ith row of the result matrix. Finally, the column means of the matrix is returned. The parameter *B* can be selected manually. In Section 4, we selected the value of 10^5 for the *B* parameter.

Algorithm 1 Generating Initial Probability Vector For a Single Variable

```
1:  procedure InitProbabilityVector
2:      $B \leftarrow$ number of samples
3:      $minval \leftarrow$ lower bound of the parameter
4:      $maxval \leftarrow$ upper bound of the parameter
5:      $bitmatrix \leftarrow B \times 32$ matrix of zeros
6:      for i = 1 ... $B$ do
7:          $value \leftarrow RandomUniform(minvals[j], maxvals[j])$
8:          $bits \leftarrow encode(value)$
9:          $bitmatrix[i,] \leftarrow bits$
10:
        end for
11:     $probs \leftarrow ColumnMeans(bitmatrix)$
12:     return probs
```

Algoritm 1. Generating Initial Probability Vector For a Single Variable

2.3. The hybrid compact genetic search

The devised method performs a genetic search using the algorithm given in Algorithm 2.3. The algorithm uses a vector of probabilities generated using the Algorithm 2.2. In each step, two candidate solutions are sampled using the probabilities. Assuming the goal function is subject to be minimized, the *winner* is the candidate chromosome with lower cost value. These parts of the genetic search are almost same with the

*CGA*s expect the *decode()* part. *decode()* receives $p \times 32$ bits as input and returns a vector of p real values which are decoded using the *IEEE-754* transformation.

Algorithm 2 Machine Coded Compact Genetic Algorithm

```
1: procedure MCCGA
2:     $f \leftarrow$ function to be minimized
3:     $probs \leftarrow$ initial probability vector prepared by InitProbabilityVector()
4:     $popsize \leftarrow$ amount of mutation
5:     $L \leftarrow$ length of bits
6:     while elements of $probs$ are not converged do
7:         $chromosome_1 \leftarrow Sample(probs)$
8:         $chromosome_2 \leftarrow Sample(probs)$
9:         $cost_1 \leftarrow f(decode(chromosome_1))$
10:        $cost_2 \leftarrow f(decode(chromosome_2))$
11:        if $f(cost_1 < cost_2)$ then
12:            $winner \leftarrow chromosome_1$
13:        else
14:            $winner \leftarrow chromosome_2$
15:        for i = 1...$L$ do
16:            if $winner[i] = 1$ && $probs[i] < 1$ then
17:                $probs[i] \leftarrow probs[i] + \frac{1}{popsize}$
18:            else if $winner[i] = 0$ && $probs[i] > 0$ then
19:                $probs[i] \leftarrow PBS[i] - \frac{1}{popsize}$
20:    return $probs$
```

Algorithm 2. Machine Coded Compact Genetic Algorithm

Note that the function *decode()* uses the single precision version of *IEEE-754* which spans 32 bits in the computer memory. The double precision version of the specification represents a wider range of numbers as it spans 64 bits. However, working with longer bit strings reduces the performance drastically.

In *GA*s, and generally in some evolutionary optimization algorithms, genetic operators perform the search by *Exploration* and *Exploitation* (Chen et al., 2009). After performing a genetic operator, a new solution can be created in a different location of the search space which covers the global optimum. On the other hand, the newly generated solution can fall a location close to the global optimum which is generated using two best solutions in the population. Shortly, the processes of *searching the new areas* and *performing local fine-tuning* are executed in parallel. The balance between these two vital tasks must be calibrated.

In some cases, a genetic search can terminate by reporting a good solution which is not the global optimum because of lack of a lucky mutation or crossover operation or overshooting due to wrongly selected adaptive probabilities. In other words, a *GA* search can find a nice solution around the global optimum and a local fine-tuning operation may be required for finding the ideal solution.

Hybridization of search algorithms is applied in several ways by combining at least two optimization algorithms. Gonçalves et al. (2015) improved the result obtained by a genetic algorithm using a local search optimization tool to prevent getting stuck on a local optimum. Kim et al (2007) combined a genetic algorithm with a particle swarm optimization tool in the run-time for searching the global optimum of multimodal functions. Arakaki and Usberti (2018) and Usberti et al. (2018) used a hybridization method based on the statistical filtering. Satman and Akadal (2016) applied *ARIMA* forecasting to predict offspring using the historical chromosome data of parents in earlier generations as a hybridization tool. Liu et al. (2018),Long and Wu (2014), Kang et al. (2011) and Satman (2015) hybridized many evolutionary optimization algorithms with the *Hooke and Jeeves* local optimizer for improving the quality and the precision of the solutions.

Hooke and Jeeves algorithm is a local search optimizer for optimization problems which are not necessarily differentiable. Algorithm starts searching using an initial solution. This initial search is modified in all directions by a predefined *step size* parameter. Successful moves are stored and the search is repeated while the decreased *step size* is not zero. The final solution is reported as the optimum (Hooke and Jeeves, 1961; Moser 2009).

In our proposed method, we apply a *Hooke and Jeeves* local search for improving the result obtain by the *CGA* defined in Algorithm 2.3. The whole algorithm is given in Algorithm 2.3.

```
Algorithm 3 Machine Coded Compact Genetic Algorithm with Hybridization
1: procedure HYBRID-MCCGA
2:     f ← function to be minimized
3:     probs ← initial probability vector prepared by InitProbabilityVector()
4:     popsize ← amount of mutation
5:     L ← length of bits
6:     initial ← MCCGA(f, probs, popsize, L)
7:     result ← Hooke-Jeeves(initial, f)
8:     return result
```

Algorithm 3. Machine Coded Compact Genetic Algorithm with Hybridization

The algorithm given in Algorithm 2.3 is mainly based on three steps. The n −variable objective function defined as $f\colon \mathcal{R}^n \to \mathcal{R}$ is transformed using the *IEEE-754* standard and redefined as $f^b\colon \mathcal{B}^{n\times 32} \to \mathcal{R}$ where $\mathcal{B}$ is the binary space. In the first stage defined in Algorithm 2.2, the initial vector of probabilities is generated. Depending on the range of the variables, some probabilities in this vector equal to 0, 1, or any value within the range. If the corresponding probability is either zero or one, the search space is divided and the remaining effort is performed on the other elements of the vector. The final probabilities are then used in the *MCCGA (Machine-coded Compact Genetic Algorithm)* stage given in the Algorithm 2.3. This stage is almost as same with the original *CGA* algorithm except the initial vector of probabilities and the *decoder* function. In this stage, the vector of probabilities is used for generating new chromosomes by sampling. The *decoder* is applied to evaluate the objective function f^b using the binary representation standard. After all of the probability elements are converged to either 0s or 1s, the stage is terminated. The final $n \times 32$ bits are decoded into the real values and a *Hooke and Jeeves* search is started. The reported solution is expected to be the global optimum.

3. An Example

The *Chichinadze* function is defined as

$$f(x,y) = x^2 - 12x + 11 + 10\cos(\pi x/2) + 8\sin(5\pi x) - \frac{1}{\sqrt{5}}e^{-0.5(y-0.5)^2}$$

for $-30 \leq x, y \leq 30$ and $f^* = -43.3159$ is the global minimum at $x = 5.90133$ and $y = 0.5$. The graphics of the function in a narrower range is shown in Figure 1.

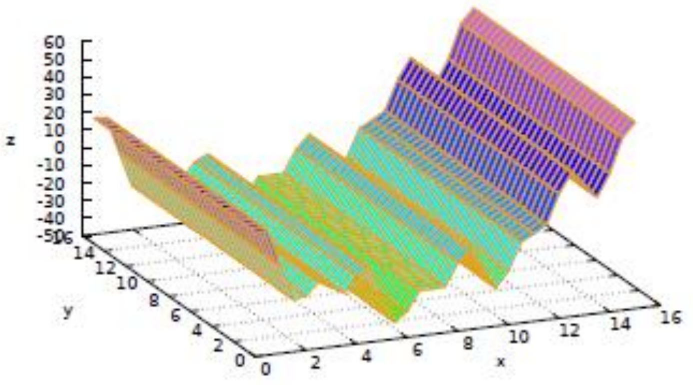

Figure 1. Chichinadze function

The classical *CGA* search with the initial vector of probabilities

$$[0.5\ 0.5\ \ldots\ 0.5]$$

reports the solution as $x = -0.0949707$, $y = 0.4999996$, and $f^{\phi} = 13.61534$ which is far from the global minimum. The *popSize* parameter is selected as 200.

Since the selected range is a considerably small zone of the whole *floating-point* representation space, some bits of the encoded variables tend to take the value of either 0 or 1 with higher probabilities. Figure 2 shows the probabilities $P(b_i = 1)$ of *IEEE-754* encoding of $-30 \leq x \leq 30$. It is shown in Figure 2 that the $P(b_i = 1)$ for $i = 3,4,5,6,7$ are under 0.2, whereas, the probabilities are above 0.6 for $i = 2,8,9$. As the range is symmetric around zero, $P(b_1 = 1)$ is calculated exactly as 0.5. $P(b_i = 1)$ for $i = 10,11,12$ are under 0.5 but the differences are negligible. A *CGA* search with the generated initial vector reports the solution as $x = 5.90625$, $y = 0.465660$, and $f^{\psi} = -43.23912$ which is closer to the global minimum. Despite the solution reported by the algorithm is satisfactory, the search capabilities can be improved. The reported solution is a good initial point for *Hooke-Jeeves* algorithm. After applying the local search process using the *CGA* based solution as the initial solution, we obtain the final solution as $x = 5.901329$, $y = 0.500000$, and $f^{\zeta} = -43.31586$.

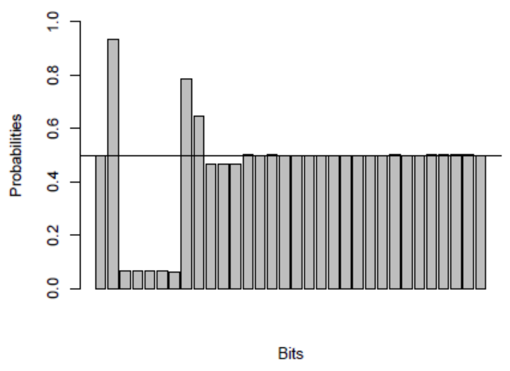

Figure 2. $P(b_i = 1)$ of *IEEE-754* encoding for $-30 \leq x \leq 30$

4. Simulations

We perform a simulation study to compare the search capabilities of *CGA* and the developed algorithm. We use a suit of test functions reported in (Mishra, 2006). This set of test functions is used to measure the performance differences of well-known optimization techniques in the literature. The simulations are repeated 1000 times for each configuration. Since the *popSize* parameter affects the performance, it is selected as 10, 20, 50, 100, and 200. For the subset of functions defined for $m \geq 2$ variables, simulations are performed for $m = 2$, $m = 10$, and $m = 25$. Simulation results for $m = 2$ are reported in Table 3-6.

The table columns represents the value of *popSize* parameter, arithmetic means and standard deviations of reported objective values by algorithms, and p-values. We applied a 2-sample *Wilcoxon Test* (*Mann-Whitney*) for independent samples to test equality of location parameters of two populations. Small p-values indicate that we can safely reject the null-hypothesis of $H_0: \mu_1 = \mu_2$ in contrast to $H_a: \mu_1 \neq \mu_2$ where μ_1 and μ_2 are location parameters of distributions related to the corresponding objective values. *NA* values are generated when the algorithms produce exaclty the same results and *NA*s can be interpreted as the p-value of 1. It is shown in Tables 3-6 that almost all of the p-values are small and this can be accepted as a general evidence for performance inequality of these methods. *CGA* outperforms the developed algorithm for *Cross leg table*, *Modified Schaffer #1*, *Modified Schaffer #2*, *Schaffer*, and *Griewank* functions. The two method performs nearly same for *Crowned cross*, *Tree humps camel back*, *Ackley*, *Bohacevsky*, *Holzman*, *Hyperellipsoid*, *Maxmod*, *Multimod*, *Rastrigin*, *Sphere*, and *Sumsquares* functions despite the reported p-values indicate the evidence of inequality of the performances. The developed algorithm outperforms the *CGA* for *Test tube holder*, *Holder table*, *Carrom table*, *Cross in tray*, *Cross*, *Pen holder*, *Bird*, *Modified Schaffer #3*, *Egg holder*, *Chichinadze*, *McCormick*, *Levy*, *Styblinski tang*, *Bukin*, *Leon*, *Giunta*, *Rosenbrock*, and *Schwefel* functions. As the value of *popSize* parameter increases the performance also increases but the difference is more apparent for the developed algorithm. Interestingly in some functions, for example *Bird* function, average performance is decreased as the *popSize* is increased but the standard deviation is also increased. That means the performance is badly affected and includes fluctuations but it reports solutions that close to the global optimum in some iterations. This exception also designates that increasing the *popSize* does not necessarily mean having a better solution reported after choosing an unlucky random seed.

Table 7-9 summarize the simulation reports for higher dimensions. The developed algorithm have better or equal performance than the *CGA* except *Sine envelope*, *Maxmod*, and *Schaffer* functions. Both of the methods fail to obtain a good solution in average for *Schwefel* function. It is also shown in results that an increase on *popSize* generally increases the quality of solutions with smaller standard deviations. But the increase in the average quality is steeper in the *CGA* especially for higher dimensions

Function	**Population Size**	**Mean of CGA**	**Mean of MCCGA**	**Std. Dev. Of CGA**	**Std. Dev. Of MCCGA**	**P-value**
TestTubeHolder (-10.8723)	10	0	-7.101	0	2.506	0
	20	-5.303	-7.415	0.855	2.600	0
	50	-5.61	-8.813	1.45	2.421	0
	100	-5.92	-9.518	1.834	1.965	0
	200	-6.663	-9.73	2.36	1.693	0
HolderTable (-26.92)	10	-2.718	-20.485	0	8.702	0
	20	-18.276	-22.677	7.454	7.646	0
	50	-20.217	-24.796	6.356	5.706	0
	100	-21.408	-25.053	5.236	5.39	0
	200	-22.155	-25.736	4.236	4.408	0
CarromTable (-24.15682)	10	-0.246	-15.748	0	10.422	0
	20	-13.02	-20.097	7.587	8.319	0
	50	-15.045	-21.672	6.423	6.737	0
	100	-16.104	-22.646	5.427	5.341	0
	200	-16.995	-23.156	4.258	4.433	0
CrossInTray (-2.062612)	10	0	-1.483	0	0.311	0
	20	-0.937	-1.377	0.517	0.26	0
	50	-1.043	-1.318	0.444	0.203	0
	100	-1.1	-1.297	0.388	0.172	0
	200	-1.186	-1.281	0.257	0.148	0
CrownedCross (0)	10	0	0	0	0.005	0
	20	0	0	0	0	0
	50	0	0	0	0	0.32
	100	0	0	0	0	NA
	200	0	0	0	0	NA
Cross (0)	10	1	0	0	0	0
	20	0.017	0	0.057	0	0
	50	0.008	0	0.038	0	0
	100	0.006	0	0.039	0	0
	200	0.004	0	0.029	0	0
CrossLegTable (-1)	10	-1	-0.996	0	0.054	0
	20	-1	-0.999	0	0.018	0
	50	-1	-1	0	0	0.32
	100	-1	-1	0	0	NA
	200	-1	-1	0	0	NA
PenHolder (-0.96354)	10	-0.692	-0.933	0	0.055	0
	20	0	-0.95	0	0.029	0
	50	0	-0.956	0	0.022	0
	100	0	-0.958	0	0.019	0
	200	0	-0.96	0	0.015	0
Bird (-106.7645)	10	2.718	-100.463	0	19.278	0
	20	2.205	-94.018	0.343	28.071	0
	50	1.872	-77.442	0.345	41.14	0
	100	1.683	-56.877	0.251	47.658	0
	200	1.586	-37.239	0.145	47.332	0
ModifiedSchaffer1 (0)	10	0	0.29	0	0.219	0
	20	0	0.307	0	0.215	0
	50	0	0.251	0	0.22	0
	100	0	0.193	0	0.213	0
	200	0	0.092	0	0.167	0
ModifiedSchaffer2 (0)	10	0	0.298	0	0.222	0
	20	0	0.322	0	0.211	0
	50	0	0.267	0	0.216	0
	100	0	0.195	0	0.215	0
	200	0	0.113	0	0.182	0

Table 3. Simulation results for $p = 2$

Function	Population Size	Mean of CGA	Mean of MCCGA	Std. Dev. of CGA	Std. Dev. of MCCGA	P-value
Modified Schaffer3 (0.00156685)	10	0.708	0.261	0	0.214	0
	20	0.436	0.189	0.153	0.206	0
	50	0.311	0.073	0.22	0.143	0
	100	0.171	0.018	0.21	0.061	0
	200	0.051	0.007	0.115	0.014	0
Modified Schaffer4 (0.292579)	10	1	0.468	0	0.066	0
	20	0.497	0.484	0.029	0.048	0
	50	0.496	0.494	0.032	0.03	0
	100	0.497	0.497	0.029	0.021	0
	200	0.498	0.498	0.024	0.014	0
EggHolder (-959.64)	10	-25.46	-562.187	0	223.319	0
	20	127.575	-652.474	431.326	207.335	0
	50	139.999	-691.437	418.092	196.009	0
	100	112.034	-732.816	374.64	180.738	0
	200	103.389	-771.031	350.95	164.206	0
Chichinadze (-43.3159)	10	20.605	-34.038	0	7.826	0
	20	15.953	-36.68	6.575	6.976	0
	50	14.591	-39.147	4.539	5.947	0
	100	14.122	-40.627	1.859	5.058	0
	200	13.795	-41.718	1.016	4.113	0
McCormick (-1.9133)	10	1	-1.913	0	0	0
	20	-0.571	-1.913	1.134	0	0
	50	-1.363	-1.913	0.615	0	0
	100	-1.567	-1.913	0.248	0	0
	200	-1.66	-1.913	0.166	0	0
Levy (0)	10	2	0.003	0	0.022	0
	20	1.625	0.001	0.459	0.008	0
	50	1.211	0.003	0.632	0.016	0
	100	0.816	0.006	0.674	0.025	0
	200	0.526	0.009	0.664	0.03	0
ThreeHumps CamelBack (0)	10	0	0	0	0.009	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0
	200	0	0	0	0	0.74
Zettle (-0.003791)	10	0	-0.004	0	0	0
	20	-0.002	-0.004	0.001	0	0
	50	-0.002	-0.004	0.001	0	0
	100	-0.002	-0.004	0.001	0	0
	200	-0.002	-0.004	0.001	0	0
StyblinskiTang (-78.332)	10	0	-77.159	0	4.15	0
	20	-28.501	-78.233	14.975	1.179	0
	50	-49.807	-78.332	12.004	0	0
	100	-56.95	-78.332	04.01.1900	0	0
	200	-57.997	-78.332	0.075	0	0
Bukin (-124.75)	10	75.25	-124.75	0	0	0
	20	141.262	-124.75	361.617	0	0
	50	100.604	-124.75	323.891	0	0
	100	75.744	-124.75	297.064	0	0
	200	55.893	-124.75	271.1	0	0

Table 4. Simulation results for p=2 (Continued)

Function	Population Size	Mean of CGA	Mean of MCCGA	Std. Dev. Of CGA	Std. Dev. Of MCCGA	P-value
Leon (0)	10	1	0	0	0	0
	20	0.91	0	0.099	0	0
	50	0.828	0	0.084	0	0
	100	0.807	0	0.066	0	0
	200	0.789	0	0.038	0	0
Giunta (0.06447047)	10	0.363	0.088	0	0.055	0
	20	0.251	0.072	0.081	0.033	0
	50	0.151	0.065	0.08	0.01	0
	100	0.104	0.064	0.057	0	0
	200	0.076	0.064	0.031	0	0
Schaffer (0)	10	0	0.315	0	0.217	0
	20	0	0.341	0	0.205	0
	50	0	0.302	0	0.213	0
	100	0	0.253	0	0.221	0
	200	0	0.201	0	0.218	0
Ackley (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	NA
	200	0	0	0	0	NA
Bohachevsky (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0.03
	200	0	0	0	0	NA
Griewank (0)	10	0	0.123	0	0.227	0
	20	0	0.051	0	0.105	0
	50	0	0.014	0	0.032	0
	100	0	0.008	0	0.005	0
	200	0	0.008	0	0.003	0
Holzman (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0
	200	0	0	0	0	0.03
Hyperellipsoid (0)	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0.01
	200	0	0	0	0	0.73
Maxmod (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0
	200	0	0	0	0	0.03
Multimod (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0
	200	0	0	0	0	0.32
Rastrigin (0)	10	0	0	0	0	0.16
	20	0	0	0	0	NA
	50	0	0	0	0	NA
	100	0	0	0	0	NA
	200	0	0	0	0	NA

Table 5. Simulation results for $p = 2$ (Continued)

Function	Population Size	Mean of CGA	Mean of MCCGA	Std. Dev. of CGA	Std. Dev. of MCCGA	P-value
Rosenbrock (0)	10	1	0	0	0	0
	20	0.903	0	0.101	0	0
	50	0.839	0	0.091	0	0
	100	0.803	0	0.06	0	0
	200	0.789	0	0.039	0	0
Schwefel (-837.9658)	10	0	-564.713	0	185.758	0
	20	46.004	-695.61	186.434	143.328	0
	50	75.887	-773.647	186.164	95.362	0
	100	105.49	-801.335	174.445	70.113	0
	200	117.696	-814.746	159.06	55.054	0
Sphere (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0
	200	0	0	0	0	0.07
Sumsquares (0)	10	0	0	0	0	0
	20	0	0	0	0	0
	50	0	0	0	0	0
	100	0	0	0	0	0
	200	0	0	0	0	0.09

Table 6. Simulation results for $p = 2$ (Continued)

Function	Population Size	Number of variables	Mean of CGA	Mean of MCCGA	Std.Dev.of CGA	Std.Dev.of MCCGA	P-value
Levy(0)	10		1.443	8.052	0	5.682	0
	50	10	1.255	0.563	0.252	0.754	0
	200		0.943	0.01	0.266	0.065	0
	10		2.805	43.545	0	11.761	0
	50	25	82.084	9.682	154.712	5.015	0.76
	200		2.576	0.467	0.271	0.539	0
Schaffer(0)	10		0	27.855	0	12.448	0
	50	10	0.016	10.555	0.231	4.572	0
	200		0	4.134	0	4.046	0
	10		0	138.307	0	26.803	0
	50	25	6.121	45.069	9.544	10.855	0
	200		0	20.157	0	7.557	0
Ackley(0)	10		0	0	0	0	0
	50	10	0	0	0	0	0
	200		0	0	0	0	0.01
	10		0	0	0	0	0
	50	25	0.167	0	1.51	0	0
	200		0	0	0	0	0
Bohachevsky(0)	10		0	0	0	0	0
	50	10	0.045	0	0.718	0	0
	200		0	0	0	0	0
	10		0	0	0	0	0
	50	25	7719.617	0	16729.91	0	0
	200		0.024	0	0.536	0	0
Griewank(0)	10		0	2.889	0	3.789	0
	50	10	0.001	0.244	0.017	0.367	0
	200		0	0.009	0	0.022	0
	10		0	0.444	0	0.759	0
	50	25	80.303	0.521	193.885	0.733	0
	200		0	0.063	0	0.114	0

Table 7. Simulation results for $p = 10$ and $p = 25$

Function	Population Size	Number of variables	Mean of CGA	Mean of MCCGA	Std.Dev.of CGA	Std.Dev.of MCCGA	P-value
Holzman (0)	10		0	0	0	0	0
	50	10	3.428	0	74.454	0	0
	200		0	0	0	0	0
	10		0	0	0	0	0
	50	25	199426.7	0	349380.2	0	0
	200		0.32	0	7.467	0	0
Hyperellipsoid (0)	10		0	0	0	0	0
	50	10	0.193	0	3.358	0	0
	200		0	0	0	0	0
	10		0	0	0	0	0
	50	25	550.005	0	885.08	0	0
	200		0.028	0	0.885	0	0
Maxmod (0)	10		0	4.485	0	4.746	0
	50	10	0.01	0.014	0.141	0.133	0
	200		0	0.004	0	0.089	0
	10		0	9.993	0	0.221	0
	50	25	4.405	2.61	4.235	4.155	0
	200		0	0.055	0	0.326	0
Multimod (0)	10		0	0	0	0	0
	50	10	0	0	0	0	NA
	200		0	0	0	0	NA
	10		0	0	0	0	NA
	50	25	0	0	0	0	NA
	200		0	0	0	0	NA
Rastrigin (0)	10		0	0	0	0	0
	50	10	0.069	0	1.101	0	0
	200		0	0	0	0	NA
	10		0	0	0	0	0
	50	25	49.673	0	77.604	0	0
	200		0.004	0	0.126	0	0
Rosenbrock (0)	10		9	0.614	0	1.44	0
	50	10	23.55	0.108	172.108	0.646	0
	200		8.865	0	0.111	0	0
	10		24	0.821	0	1.613	0
	50	25	2235399	0.482	3334847	1.301	0
	200		751517.7	0.024	2119229	0.308	0
Schwefel (-189.829, -10474.5725)	10		0	-2380.959	0	446.686	0
	50	10	-5.881	-3330.335	412.897	446.297	0
	200		16.142	-3910.476	409.475	249.125	0
	10		0	-5428.361	0	695.657	0
	50	25	-13.716	-7074.333	641.135	1038.013	0
	200		-0.213	-8801.598	650.614	1050.941	0
Sphere (0)	10		0	0	0	0	0
	50	10	0.226	0	4.481	0	0
	200		0	0	0	0	0
	10		0	0	0	0	0
	50	25	122.937	0	219.366	0	0
	200		0.008	0	0.179	0	0

Table 8. Simulation results for $p = 10$ and $p = 25$ (continued)

Function	Population Size	Number of variables	Mean of CGA	Mean of MCCGA	Std.Dev.of CGA	Std.Dev.of MCCGA	P-value
Sumsquares (0)	10		0	0	0	0	0
	50	10	0.266	0	6.485	0	0
	200		0	0	0	0	0
	10		0	0	0	0	0
	50	25	1814.545	0	3363.846	0	0
	200		0.096	0	3.036	0	0
SineEnvelope (0)	10		0	2.783	0	0.829	0
	50	10	0	2.272	0	0.838	0
	200		0	0.783	0	0.597	0
	10		0	8.179	0	1.248	0
	50	25	0	6.995	0	1.318	0
	200		0	3.123	0	1.212	0

Table 9. Simulation results for $p = 10$ and $p = 25$ (continued)

5. Conclusion

Each single bit of an IEEE-754 encoded real value has a different impact depending on the location of the bit. As a result of this, some bits tend to be zero or one when a variable defined in a narrower range. Process of assigning 0.5s to elements of the initial vector of probabilities in CGAs does not consider these biases. In this paper we suggest to generate the initial vector of probabilities depending on the location of bits encoded by the 32-bits IEEE-754 standard. This special binary coding scheme is used elsewhere before and proved to be success in many works. In the second stage of the extension, a usual CGA search is applied on the objective function using the same encoding scheme. In order to improve the solutions obtained by CGA, Hooke-Jeeves algorithm is applied using the reported result as the starting point. An other local search method can be used instead, however, Hooke-Jeeves algorithm has many benefits including applicability in non-differentiable functions. When a good starting point is fed, the algorithm performs a fine-tuning operation to obtain a closer solution to the global optimum. We perform a simulation study using a set of well-known test functions to measure the performance differences. Simulation results show that the hybridized and machine-coded CGAs outperform the classical CGAs.

References

Aporntewan C. and Chongstitvatana P. (2001) A hardware implementation of the compact genetic algorithm. In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, volume 1, pages 624–629. IEEE, 2001.

Arakaki, R. K and Usberti, F. L. (2018) Hybrid genetic algorithm for the open capacitated arc routing problem. Computers & Operations Research, 90:221–231.

Budin, L., Golub, M., & Budin, A. (2010). Traditional techniques of genetic algorithms applied to floating-point chromosome representations. Sign, 1(11), 52.

Chen, J., Xin, B., Peng, Z. Dou, L. and Zhang, J. (2009) Optimal contraction theorem for exploration–exploitation tradeoff in search and optimization. IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, 39(3), 680–691.

Goldberg, D. E. (1991) Real-coded genetic algorithms, virtual alphabets, and blocking. Complex systems, 5(2). 139–167.

Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning.

Goldberg. D. E (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Gonçalves, J. F. And Mendes, J. J. M and Resende, M. GC. (2005) A hybrid genetic algorithm for the job shop scheduling problem. European journal of operational research, 167(1), 77–95.

Harik, G. R., Lobo, F. G. and Goldberg, D. E. (1999) The compact genetic algorithm. IEEE transactions on evolutionary computation, 3(4). 287–297.

Holland, J. H. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

Hooke, R. and Jeeves, T. A. (1961) “direct search” solution of numerical and statistical problems. Journal of the ACM (JACM), 8(2), 212–229.

Ieee standard for floating-point arithmetic. IEEE Std 754-2008, pages 1–70, Aug 2008.

Kang, F., Li, J., Ma, Z., & Li, H. (2011). Artificial bee colony algorithm with local search for numerical optimization. Journal of Software, 6(3), 490-497.

Kim, D. H., Abraham, A. and Cho, J. H. (2007) A hybrid genetic algorithm and bacterial foraging approach for global optimization. Information Sciences, 177(18), 3918–3937.

Liu, D., Liu, C., Zhang, C., Xu, C., Du, Z. and Wan, Z. (2018), "Efficient hybrid algorithms to solve mixed discrete-continuous optimization problems: A comparative study", Engineering Computations, 35(2), 979-1002.

Long, Q., & Wu, C. (2014). A hybrid method combining genetic algorithm and Hooke-Jeeves method for constrained global optimization. Journal of industrial and management optimization, 10(4), 1279-1296.

Mininno, E., Cupertino, F., & Naso, D. (2008). Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions on Evolutionary Computation, 12(2), 203-219.

Mishra, S. K. (2006) Some new test functions for global optimization and performance of repulsive particle swarm method. Available at SSRN 926132.

Moser, I. (2009) Hooke-jeeves revisited. In Evolutionary Computation, 2009. CEC'09. IEEE Congress on, pages 2670–2676.

Ojha, V. K., Dutta, P., Saha, H., & Ghosh, S. (2012). Application of real valued neuro genetic algorithm in detection of components present in manhole gas mixture. In Advances in Computer Science, Engineering & Applications, 333-340. Springer, Berlin, Heidelberg.

Pelikan, M., Hauschild, M. W., & Lobo, F. G. (2015). Estimation of distribution algorithms. In Springer Handbook of Computational Intelligence (pp. 899-928). Springer, Berlin, Heidelberg. Baluja, S. (1994). Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science.

Sastry, K., Goldberg, D. E and Kendall, G. (2014). Genetic algorithms. In Search methodologies, Springer, 93–117.

Satman, M. H. (2013), Machine coded genetic algorithms for real parameter optimization problems. Gazi University Journal of Science, 26(1), 85–95.

Satman, M. H. (2015) Hybridization of floating-point genetic algorithms using hooke-jeeves algorithm as an intelligent mutation operator. Journal of Mathematical and Computational Science, 5(3), 320-332.

Satman, M. H. and Akadal, E. (2016) Arima forecasting as a genetic inheritance operator in floating-point genetic algorithms. Journal of Mathematical and Computational Science, 6(3), 360.

Satman, M. H. and Akadal, E. (2017) Machine-coded genetic operators and their performances in floating-point genetic algorithms. International Journal of Advanced Mathematical Sciences, 5(1), 8–19.

Sebag, M. And Ducoulombier, A (1998). Extending population-based incremental learning to continuous search spaces. In International Conference on Parallel Problem Solving from Nature, pages 418–427. Springer,

Umbarkar, A. J., Joshi, M. S., & Sheth, P. D. (2015). Dual population genetic algorithm for solving constrained optimization problems. International Journal of Intelligent Systems and Applications, 7(2), 34.

Usberti, F. L., França, P. M. and França, A. L. M. (2013) Grasp with evolutionary path-relinking for the capacitated arc routing problem. Computers & Operations Research, 40(12), 3206–3217.