An Introduction to a Novel Crossover Operator for Real-Value Encoded Genetic Algorithm: Gaussian Crossover Operator

Michał Kubicki, Daniel Figurowski
Faculty of Electrical Engineering
West Pomeranian University of Technology
Szczecin, Poland
e-mail: {michal.kubicki, daniel.figurowski}@zut.edu.pl

***Abstract*—The following article describes a novel implementation of a crossover operator for real-value encoded Genetic Algorithms (GA). The method, Gaussian Crossover Operator (GCO), utilizes the properties of Gaussian functions and Gaussian distribution for offspring generation. Each parent's fitness is evaluated in the context of general population by a heuristic function, i.e. the devised operator is performance based – the parents' individual fitness values act as a basis for a non-deterministic weighing mechanism. The child's gene value is a Gaussian Variable drawn upon the normal distribution determined by the overall state of the algorithm and the antecedent's evaluation.**

The performance of the algorithm is discussed and compared with the underlaying classical Genetic Algorithm and other GA implementations found in the literature; several test cases are considered. The results show that the proposed Gaussian Crossover Operator is feasible for solving optimization problems.

Keywords—crossover operator, gaussian, normal distribution, value-based encoding, genetic algorithm

I. INTRODUCTION

Genetic Algorithm (GA) is one of the many nature-inspired metaheuristic algorithms used extensively in optimization problems. They attempt to imitate the evolution process that governs the gradual adjustment of entire species to the environment they inhabit. As its nature counterpart, the algorithm may span across multiple generations not necessarily yielding an optimal solution but a sufficiently good one.

The many problems in which Genetic Algorithms are employed include: timetabling, system identification, device design and routing. Each of these problems may demand a different approach to the way in which a particular instance of the Genetic Algorithm is designed. This diversity led to numerous variations in how the solutions are encoded and how the three core operators (i.e. selection, crossover, mutation) are implemented.

The authors propose a new approach to one of these operators the crossover operator. The aim was the creation of a more diversified offspring population even in the case of a stale gene pool, thus enhancing the performance especially in the later iterations of GA. Also the method should aid the algorithm while searching for the extremes. This was achieved by applying an operator based upon the normal distribution and underlaying Gaussian functions and their properties, thus the name: Gaussian Crossover Operator.

II. METHOD DESCRIPTION

A. General Remarks

The method relies heavily upon the product of Gaussian functions (1) and the fact that the result is also a Gaussian function.

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{1}$$

The probability density function (PDF) of normal distribution (also called Gaussian distribution) (2) is structurally related to the Gaussian functions as can be seen when comparing these formulas.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2}$$

where μ – mean, σ^2 – variance.

However, the result of Gaussian PDF multiplication may be considered a Gaussian PDF only after a scaling factor is applied. This is due to the normalization of the resulting distribution. A more detailed mathematical description of these properties is provided in [1,2]; another important operation is the truncation of a continuous normal distribution to specific boundaries [3].

Before delving into the specifics of the designed operator it is necessary to briefly outline the properties of underlying Genetic Algorithm. The algorithm belongs to the family of real-coded genetic algorithms (RCGA) [4] – each gene is represented by a single floating point number. The selection of parents for the crossover utilizes the roulette method, same pairs of parents may be chosen multiple times. Elitism is used to preserve the best individuals from being distorted either by mutation or the non-deterministic nature of the crossover operator. Mutation is based on uniform distribution with two mutation probabilities. The first one checks whether a child should be mutated at all. If

978-1-5386-6143-7/18/$31.00 ©2018 IEEE

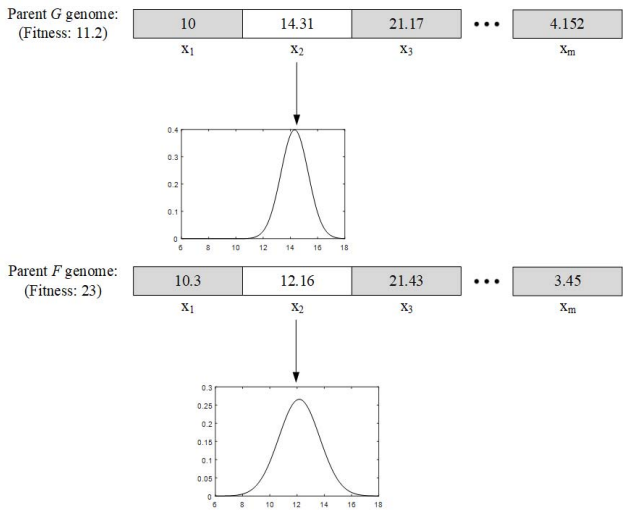

Fig. 1. Creation of x_2 gene PDFs for each of the parents. The mean (μ) is based on the parent's gene value while the variance depends on the Σ-function evaluation.

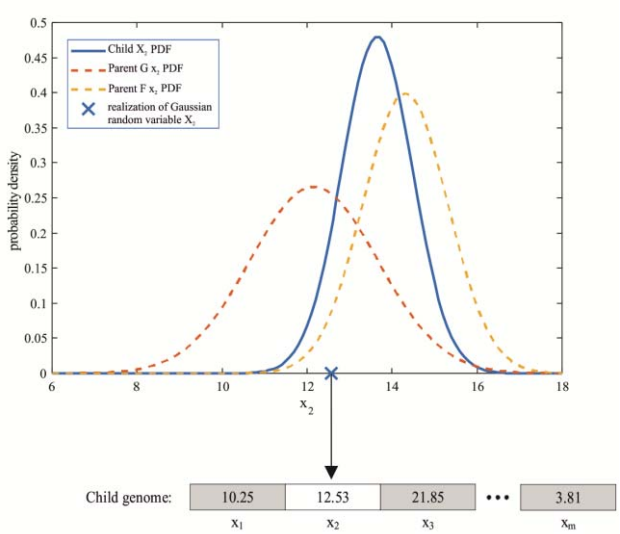

Fig. 2. Child's x_2 gene PDF is a result of multiplication of its parents' distributions. After the child's normal distribution is calculated a realization of Gaussian random variable X_2 is drawn – the value is assigned to its respective place in the genome.

so, the second one decides the probability of mutation occurring in each gene (i.e. each gene is decided separately).

B. Notation

Each individual will be denoted as P_k with k representing its unique identifier in the population. A set POP is defined to represent the entirety of the population comprised of n individuals:

$$POP = \{P_1, P_2, P_3, \dots, P_k, \dots, P_n\}. \tag{3}$$

Each individual P has one chromosome that contains the information kept in m genes (4). Each gene x stores real-encoded value that represents a part of the solution.

$$P = \{x_1, x_2, \dots, x_i, \dots, x_m\} \tag{4}$$

The values stored in the genes can be unbounded, i.e. defined in the entirety of $\mathbb{R}$, or bounded and thus limited to a specific range in $\mathbb{R}$. Possible values of gene x_i constitute its search space S_i.

It is also useful to introduce a notation that will include information about the gene index (i.e. locus) and the individual it belongs to.

$$P_k = \{x_{k,1}, x_{k,2}, \dots, x_{k,i}, \dots, x_{k,m}\} \tag{5}$$

is an example of extending the more general form (4) using this convention. Subscript used to describe gene values informs about the indices of parent (k) and gene (i) in population and chromosome respectively.

Fitness function is denoted in uppercase F, while the value it returns uses the lowercase. The evaluation of k-th individual (i.e. its fitness value) is simply

$$f_k = F(P_k). \tag{6}$$

C. The Algorithm

In the proposed method the values of genes are treated as a normal distribution. The value of the gene determines the mean (μ), while the variance (σ) can be interpreted as an algorithm's confidence in this value. The function that determines the parent's σ value will be called Σ-function or *big-sigma function*

$$\Sigma_{k,i} = \Theta_i \Psi_k\big(0.5 + U(0,1)\big). \tag{7}$$

The overall crossover operator logic in an unbounded problem is as follows:

1. For each parent's gene a PDF function is defined with μ as its current value and σ determined by the Σ-function. (Fig. 1).
2. For each matching gene pair, i.e. same index in parents' chromosomes, a Gaussian product is calculated – as a result child's PDF is achieved (Fig. 2).
3. A realization of Gaussian random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ is drawn based upon the distribution achieved in the previous stage – it becomes the child's gene value (Fig. 2).

For a bounded problem with boundaries a, b the PDF achieved in the 2nd stage is converted to a truncated version [3]. In this particular case, when the variances are large, the normal distribution resembles a uniform one. This has profound consequences for the Σ-function that will be discussed later.

Even though the mean of child's distribution is bounded by the means of its parents, the gene value is not necessarily so. The variable is drawn from the entirety of the i-th gene search space (S_i). This is demonstrated in Fig. 2 by the x_3 gene value, the outcome falls outside the range $[\mu_{G,3}, \mu_{F,3}]$.

The Σ-function (7) comprises two components. One is performance related (Ψ_k), evaluating the performance of the k-th

individual across the entire population; it serves as an weighing factor. The other (Θ_i) considers the general state of the algorithm and statistical properties of values held in genes at i-th locus. Its purpose is twofold, firstly, it serves as a scaling factor for the variance and, secondly, it monitors the overall state of the algorithm counteracting its stalling.

$$f_{avg} = F_{avg}(POP) = \frac{1}{n}\sum_{k=1}^{n} f_k \tag{8}$$

$$f_{min} = F_{min}(POP) = \min(f_1, f_2, \dots, f_n) \tag{9}$$

As mentioned before, the weighing of each parent's performance is done in context of the entire population. The metrics defined to aid in this process are: average fitness value of the population (8) and minimum fitness value found in the population (9). The function itself is a modified logistical function normalized to return values from the range (0.5,1.5):

$$\Psi_k(f_k, f_{min}, f_{avg}) = \frac{2}{1 + e^{\frac{f_k - f_{min}}{-f_{avg}}}} - 0.5. \tag{10}$$

The minimal value of Ψ (and consequently Σ) is obtained when the parent is the best fitted individual in the population.

Another important aspect is the Θ-function (11) that gradually increases variance for a specific gene when consecutive stall iterations occur (i.e. the overall population fitness is largely unchanging). Its value is decided by picking the lesser value of the two parameters:

$$\Theta_i = \min\left(\gamma\sigma_i, \frac{|S_i|}{6}\right) \tag{11}$$

where σ_i is the standard deviation for genes located at i-th locus across the entire population and $|S_i|$ is the length of the search space.

The γ parameter (12) is increasing with each consecutive stale iteration (g_{stall}) after a threshold is met, it is reset whenever a better solution is found that overcomes the old one by some error margin.

$$\gamma = \begin{cases} 1.001^{(g_{stall}-5)} & , if\ g_{stall} - 5 \geq 0 \\ 1 & , otherwise \end{cases} \tag{12}$$

As γ can rise infinitely, the second parameter of Θ-function serves as a fallback value. It limits the influence of γ to a part of the gene's search space, thus preventing the algorithm from behaving in a completely random fashion.

The Θ-function serves as a mean to introduce adaptability mechanism to the algorithm. Adaptive Genetic Algorithm (AGA) [5] adjusts its parameters (e.g. crossover rate, mutation rate, mutation strength) in accordance with the algorithm's overall state. In the devised method this is controlled by the γ parameter, increase in its value leads to higher Θ-function values. Consequently, this yields higher σ values and thus a gradual conversion of gaussian distribution to a uniform one can be observed. This reverses the trend of relying on the genetic

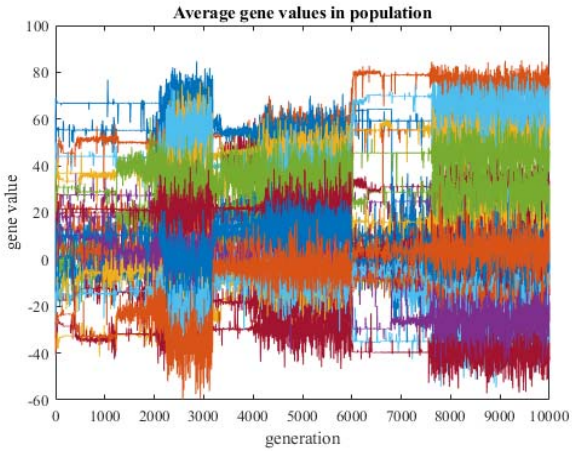

Fig. 3. Illustration of Θ pressure on the ability of the GA to traverse the search space. As new, better gene values were found the pressure was removed ($\gamma = 1$) and a more calm approach adopted to browse the search space near the newly found minimum.

information already stored in the population and leads to a more adventurous exploration of the search space.

The effect of Θ-function is clearly visible in Fig. 3. The gradual increase in noisiness is attributable to increasing value of γ parameter as consecutive algorithm iterations show no improvement. However, in time this growth is halted. This in turn is the effect of threshold put on the Θ-function by the factor tied to the search space length. After a new solution was found, the γ parameter was reset resuming the normal operation of algorithm. It is noteworthy that the entire process occurred several times during single execution of the algorithm.

Finally, an additional randomizing factor was introduced in (7). The goal of that uniform distribution was to further increase the diversity of children spawned by the same parent pair.

D. Classifying the Algorithm

Out of many crossover operators presented in [6] dedicated for real-value encoded Genetic Algorithms, some deserve special attention in context of the presented method. In [6] a taxonomy for organizing the crossover operators was proposed and of particular interest is the family of neighborhood-based crossover operators (NBCO). The main difference between the proposed methods and other NBCOs (SBX, FR, BLX-a) is not only the shape of their probabilistic functions but also the way in which offspring genes are determined, the focus is centered on the parents' genes.

In [7] the author discusses different approaches to selection of the search regions while also providing insight into how some of the previously mentioned crossover operators work. This discussion also includes the BNDX and UNDX [8] operators; the former one is parent-focused, while the latter mean-focused. An analysis is made in [9] on striking the balance between these two approaches; as a compromise the author proposes ANDX method [7,9].

The proposed algorithm can be understood as a hybrid solution that employs a heuristic Σ-function for mean determination but also behaves as a mutation operator in close vicinity to the mean.

TABLE I. PERFORMANCE COMPARISON OF CORE GA AND GCO

	GA_CORE			GA_GCO		
Test function	**Best**	**Average**	**Worst**	**Best**	**Average**	**Worst**
F1* (Sphere) [10]	0*	0*	0*	0*	0*	0*
F2 (Axis paralel Hyper-ellipsoid) [10]	0*	0*	0*	0*	0*	0*
F3 (Rotated Hyper-Ellipsoid Function) [10]	542.0973	1.8893e+03	5.1317e+03	4.6137e-09	3.1186e-07	1.4290e-06
F4 (Schwefel) [10]	-1.1577e+04	-1.1099e+04	-1.0446e+04	-1.1977e+04	-1.1397e+04	-1.0258e+04
F5 (Rastrigin) [10]	0	0	0	0	1.2710	21.6051
F6 (Rosenbrock) [10]	26.1034	28.3733	29.5263	25.0765	28.2775	28.8703
F12 (Ackley) [11]	1.5099e-14	2.2441e-14	2.9310e-14	4.4409e-15	7.1645e-15	7.9936e-15
F13 (Fifth function of De Jong) [11]	52.2022	83.0898	120.3298	51.2591	99.1676	142.5031
F14 (Modified Schaffer #1) [12]	0.0437	0.1202	0.4309	4.3012	4.8784	5.3730
F15 (Modified Schaffer #4) [12]	4.4097	4.4670	4.6196	6.4378	6.6399	6.8397
F16 (Cross-leg table function) [12]	-15	-14.0669	-2.0838	-1.0280	-0.2112	-0.0016

*Floating point precision reached

III. COMPARISON WITH THE CORE-GA

In order to show the efficiency of the Gaussian Crossover Operator it was compared to the GA implementation it was based upon. The main difference is the crossover operator - base algorithm uses modified constant arithmetical crossover similar to the one found in [4]. The caveats being that only one child is generated, each gene has its own λ and the crossover allowed for multiple parents. For only two parents it translates simply to:

$$h_i = \lambda_i x_{1,i} + (1 - \lambda_i) x_{2,i} \tag{13}$$

where h_i – child's i-th gene value, $x_{k,i}$ – k-th parent i-th gene value, λ_i-variable drawn for i-th gene from $U(0,1)$.

Some of the tests were intentionally chosen to overlap with ones found in next section. The goal of this comparison is to discern the strength of the underlying core algorithm with and without the proposed operator.

Both core algorithm and its GCO extension used the same set of parameters without fine-tuning them; each algorithm was run 30 times. Dimensionality (D) of each problem was set to 30, population size (n) to 20, number of iterations set to 10^5 (i.e. no premature stop criterion). The populations were identical in both cases as the same seed was used for their generation. Whenever a floating point precision value was reached an according remark is provided.

The discussion of the benchmark functions will be omitted as their description is often provided by the referenced work; only their name and reference will be given. Boundaries match those found in referenced work as well as typical parameters.

Results in Table I show notable performance differences between algorithms, further exemplified in Fig. 4 and Fig. 5. For further discussion see Section V.

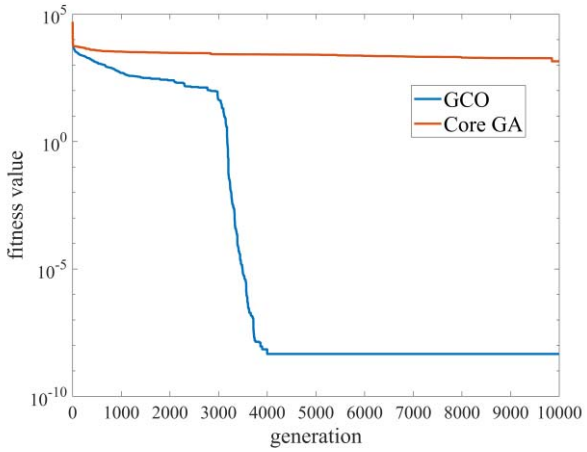

Fig. 4. Fitness value achieved by the GCO and core algorithm for 10000 iterations for test function F3.

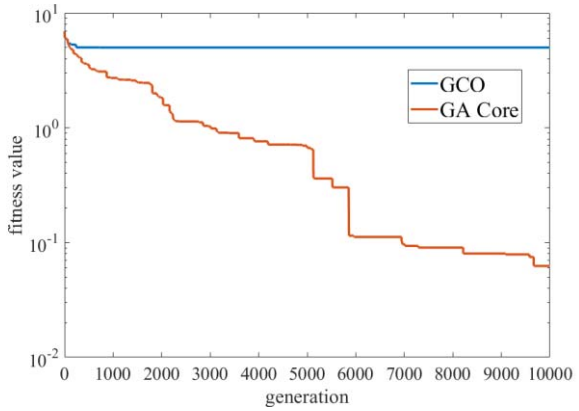

Fig. 5. Fitness history for F14 test function; a different outcome – the core algorithm outperforms the new operator.

TABLE II. COMPARISON WITH THE RESULTS PRESENTED BY KAYA ET AL. [10]

	Kaya et al. [10]			GA_GCO			GA_CORE
Test function	**Best**	**Average**	**Worst**	**Best**	**Average**	**Worst**	**Average**
F1 (Sphere)	0.0027 (RC)	0.3299 (RC)	6.163 (RC)	**7.5449e-10**	3.5381e-08	**1.7380e-07**	**4.4212e-08**
F2 (Axis Parallel Hyper Ellipsoid Function)	0.024 (HC)	5.706 (HC)	80.4 (IC)	**4.6653e-08**	**8.5077e-07**	**5.6467e-06**	4.8261e-07
F3 (Rotated Hyper-Ellipsoid Function)	**2.36** (HC)	**18.97** (RC)	**47.94** (IC)	321.7098	1.1429e+03	2.4744e+03	4.2747e+03
F4 (Normalized Schwefel Function)	-117.8 (RC)	-117.7 (RC)	-29.46 (SPC)	**-8.8393e+03**	**-6.9396e+03**	**-4.6185e+03**	-6.0527e+03
F5 (Generalized Rastrigin Function)	**2.669 (RC)**	**3.691 (RC)**	173.1 (HC)	4.2491	20.4432	**40.1336**	5.8685
F6 (Rosenbrock's Valley Function)	27.08 (AC)	**27.12 (AC)**	260.3 (AC)	**26.3116**	28.7471	**30.0513**	28.7201

*Floating point precision reached

IV. COMPARISON WITH OTHER METHODS

The results achieved by the devised method were also compared to the outcomes of other researchers who used their own solutions for a range of test problems. However, it is also crucial to note the fact that the algorithms presented in this section differ greatly. A novel binary encoding is used in [10] while *Charibde* [13,14] is not a GA at all. The comparison serves as a mean to prove the feasibility of GCO in solving optimization problems.

In [10] authors compare their crossover operator (Ring Crossover) with other crossover methods. While the most relevant parameters (e.g. problem dimensionality, population size, maximum number of iterations, boundaries) were left the same, there are some differences that are more or less specific to the devised algorithm (e.g. mutation rate, selection method). The stop criterion was defined to be 10000 evaluations (i.e. fitness function evaluations), which translates to 555 iterations of the proposed algorithm.

Table II contains the results achieved by GCO, its core algorithm and binary-encoded crossovers presented in [10]. In the latter case the result of best performing crossover operator is provided, not necessarily the Ring Crossover (RC) developed by the authors of the study. For reference of used abbreviations and test functions the reader is encouraged to read the original paper.

In [13] authors confront their state-of-the-art solution to various solvers available in the NEOS server [15]. Similarly to the previous case, the GCO is compared to these results (Table III). This comparison, however, combines the approach taken in the previous ones, i.e. Core-GA and [10]. Twofold tests were run: fixed number of iterations (10000) and iterations limited by the calls to an evaluation function. This latter boundary was tied to the number of evaluations made by the PGAPack algorithm available in the NEOS server whose results were also presented in [13]. Each test function yielded different number of evaluations but on average it correlates to about 540 iterations run by the GCO. Also in order to remain consistent with PGAPack results, algorithm was run 5 times and only the best result is presented.

As this study provides the time that was used by the algorithm to process a test function, the same was done for GCO. The simulations were conducted with the use of Matlab 2017b package on Lenovo Thinkpad X240 (CPU: Intel Core i7-4600U 3.3 GHz, RAM: 8GB DDR3 1600 MHz).

The test set in [13] is also quite different from the one found in [10].All of functions have a non-trivial global minimum. Data for parameters a_{ij} and c_i for Shekhel Foxhole Problem is assumed to be consistent with that found in [16]. The Keane function was not considered by the GCO as it is a constrained optimization problem.

TABLE III. COMPARISON WITH THE RESULTS PRESENTED BY VANARET ET AL. [13]

	Vanaret et al. [13]			GA_GCO				
Test function	**Min. Value**	**Time**	**PGAPack**	**Best**		**Average**	**Worst**	**Avg. run time**
				10000 iterations	**PGAPack limited**			
F7 (Michalewicz)	-49.6248323	8.4s	-37.60465	-35.480269	-31.434413	-29.986504	-27.179425	17.04s
F8 (Sine Envelope)	-5.9659811	219s	-5.569554	-5.9659811	-5.965976	-5.9659390	-5.9656547	6.12s
F9 (Shekel)	-10.4039521	0.04s	-1.829452	-3.294076	-3.294076	-2.299682	-1.833764	7.31s
F10 (Egg Holder)	-3719.7248363	0.8s	-3010.073	-3516.03687	-2879.7667	-2748.33093	-2069.196701	7.11s
F11 (Rana)	-2046.8320657	17.8s	*-2091.068*	-1924.57884	-1607.9953	-1604.29319	-1258.119689	7.96s

V. Discussion

The devised operator outperforms the core algorithm clearly for most of the simple problems (Fig. 4), especially convex functions and albeit not shown in this paper, it also works better for test problems with non-trivial minima [13]. However, it also has problems with getting stuck in local minima and in those cases the base version tends to outperform it (Fig. 4). Adjustments to the Θ-function and redesigning the mutation operator in order to specialize it for more aggressive activity in farther search space should be sufficient.

It is also worth noting that the way those two algorithms operate is very different. Core GA has more abrupt shifts in fitness value while the GCO tends to generate a smoother path. Even though Fig. 4 looks like a cliff, the descent is made across hundreds of generations. Another proof of this behavior is visible when comparing data from Table I and Table II. In the first case, there was virtually no constraint put on the algorithms' resources – thousands of iterations were given to each of them. However, when applying test problems from [10], a serious computational restraint was put in place. The fitness values achieved by the GCO when ended much earlier are far from those presented in Table II; the most evident example is the F3 test problem.

As for the comparison with other methods, the GCO performed satisfactorily in test set presented in [10] in context of the results achieved by binary-encoded GAs. In no way should those results reignite the historical debate between RCGA and BCGA (binary coded genetic algorithm) [17,18]. Obviously, while performing the test set of [13] the GCO was no match to the state-of-the-art *Charibde* solver. However, a valuable insight into its overall performance is given by its GA counterpart – PGAPack. The results achieved by both these algorithms are somewhat comparable as shown in Table III.

VI. Further Work

The further refinement of the proposed method is an ongoing process and some of the new developments will be published in a future paper. Some of the current research topics regarding the developed algorithm include:

- Development of a mutation operator that fully synergizes with the concept of the presented method.
- Enabling the use of constraints in the optimization process.
- Different types of Σ-function and γ-function, analysis of their influence on the algorithm's performance.
- Multiparent crossover.
- Consideration of epistasis and introduction of covariance in the algorithm.
- Application of the algorithm to a real-world problem (i.e. robot path planning).

Also more focus will be given to some of the more intrinsic mechanisms implemented in the algorithm. Finally, a more detailed algorithm's performance evaluation will be provided using recent CEC test suite and comparing the results to its counterparts.

References

[1] P. A. Bromiley, "Products and convolutions of Gaussian probability density functions," Tina memo no. 2003-003, internal report, 2014.

[2] J. Wu, "Some properties of the normal distribution," Nanjing University, China, 2017.

[3] J. Burkardt, "The truncated normal distribution." Florida State University, 2014.

[4] F. Herrera, M. Lozano, and J. L. Verdegay, "Tackling real-coded genetic algorithms: operators and tools for behavioural analysis," Artificial Intelligence Review, vol. 12, no. 4, pp. 265–319, 1998.

[5] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, no. 4, pp. 656–667, 1994.

[6] F. Herrera, M. Lozano, and A. M. Sánchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: an experimental study," International Journal of Intelligent Systems, vol. 18, no. 3, pp. 309–338, 2003.

[7] H. Someya, "Promising search regions of crossover operators for function optimization," in New Trends in Applied Artificial Intelligence, 2007, pp. 434–443.

[8] I. Ono, H. Kita, and S. Kobayashi, "A real-coded genetic algorithm using the unimodal normal distribution crossover," in Advances in Evolutionary Computing, 2003, pp. 213–237.

[9] H. Someya, "Striking a mean- and parent-centric balance in real-valued crossover operators," IEEE Transactions on Evolutionary Computation, vol. 17, no. 6, pp. 737–754, 2013.

[10] Y. Kaya, M. Uyar, and R. Tekin, "A novel crossover operator for genetic algorithms: ring crossover," Computing Research Repository, 2011.

[11] M. Molga and C. Smutnicki, "Test functions for optimization needs," Computer Information Science, 2005.

[12] S. K. Mishra, "Some new test functions for global optimization and performance of repulsive particle swarm method," Social Science Research Network, 2006.

[13] C. Vanaret, J.-B. Gotteland, N. Durand, and J.-M. Alliot, "Certified global minima for a benchmark of difficult optimization problems," Genetic and Evolutionary Computation Conference, 2014.

[14] C. Vanaret, J.-B. Gotteland, N. Durand, and J.-M. Alliot, "Hybridization of interval CP and evolutionary algorithms for optimizing difficult problems," in Principles and Practice of Constraint Programming, 2015, pp. 446–462.

[15] W. Gropp and J. J. Moré, "Optimization environments and the NEOS server," in Approximation Theory and Optimization, 1997, pp. 167-182.

[16] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems," Journal of Global Optimization, vol. 31, no. 4, pp. 635–672, 2005.

[17] C. Z. Janikow and Z. Michalewicz, "An experimental comparison of binary and floating point representations in genetic algorithms," International Conference on Genetic Algorithms, 2002, pp. 31–36.

[18] J. H. Holland, "Adaptation in natural and artificial systems," University of Michigan Press, Ann Arbor, 1975.