Available online at http://scik.org
J. Math. Comput. Sci. 5 (2015), No. 3, 320-332
ISSN: 1927-5307

HYBRIDIZATION OF FLOATING-POINT GENETIC ALGORITHMS USING HOOKE-JEEVES ALGORITHM AS AN INTELLIGENT MUTATION OPERATOR

MEHMET HAKAN SATMAN

Department of Econometrics, Istanbul University, Istanbul, Turkey

Abstract. Hybridization of genetic algorithms increases the search capabilities by means of convergence rate and speed. In this paper, we suggest to use Hooke-Jeeves algorithm as a genetic operator which performs a local search using the best chromosome in a generation as the base point. As Hooke-Jeeves algorithm searches a subspace in all directions of parameters for a given starting point, it can be considered as an intelligent mutation operator, whereas, the classical mutation operator is totally blind. The operator is applied within a predefined probability. Simulation studies performed on optimizing some well-known set of test functions show that using such an intelligent mutation operator has significant effects even for small number of iterations.

Keywords: Genetic algorithms; Local search; Hybridization.

2010 AMS Subject Classification: 65K10, 97R40, 65C05.

1. Introduction

Genetic algorithms (GAs) are parallel search and optimization methods which mimic the principles of natural selection and genetic processes [1, 2]. GAs do not require the goal function to be neither differentiable nor continuous over the search space. It is shown that the GAs outperform classical optimization methods depending on the correct selection of some genetic operators and parameters including population size, type of selection, type of crossing-over,

Received February 15, 2015

probabilities of crossing-over and mutation among others [3, 4]. Since GAs have great success on finding the global optimum, this process heavily depends on the diversity of population, that is, initially randomized population may not include the enough information to reach the global optimum by applying well-known genetic operators such as selection, crossing-over, mutation and elitism.

As a result of genetic search the algorithm may

- find the global optimum
- converge by finding a good solution near the global optimum
- get stuck on a local optimum.

In the first case of situations above, the algorithm is said to be success. In the second case, the solution is no more improved by genetic operators. Hybrid Genetic Algorithms (HGAs) are another members of GAs which are based on applying a local search algorithm after a genetic search and/or optimizing the parameters of optimizer and then applying a genetic search. Since there are many ways of hybridizing GAs, the main idea underlying the hybridization is to combine an external optimizer with GAs to improve the performance of algorithm in terms of reaching the global optimum or speed. HGAs are also used for the problem given in the last situation to improve search capabilities [5].

In this paper, we suggest using a new operator for floating-point GAs. The operator is applied within a pre-defined probability on the best solution in a generation. This operator is basically based on modifying the best solution using a local-search algorithm and transferring the generated offspring into the next generation with its parent to maintain genetic diversity and/or fine-tuning. It is shown that the devised operator is an *intelligent mutation* operator and mutating all parameters in right directions is better than the usual mutating operators which are applied using a given probability.

In Section 2, we refer floating-point GAs and the operators used in the proposed operator. In Section 3, we present the new operator and its extensions. In Section 4, we perform a simulation study using a well-known set of test functions with several numbers of independent variables. Finally we conclude in Section 5.

2. Floating-point genetic algorithms

Classical GAs perform a genetic search using a population of binary-coded candidate solutions to optimize a goal function over a search space. After applying genetic operators such as selection, crossing-over, mutation, elitism, etc., the members of newly generated populations are expected to have better quality on optimizing the goal function over iterations.

Floating-point GAs (FPGAs) form an other type of GAs in which the parameters are directly coded in machine floating-point numbers and they have their special types of crossing-over and mutation operators. Since the phenotype-genotype distinction is not necessary in most of FPGAs, MCGAs (Machine Coded Genetic Algorithms) use the byte representation of numbers as genotype and apply genetic operators in a way similar to classical GAs [6]. [7] stated that chromosomes coded in lower cardinality alphabets enclose much information about the search space, on the contrary, FPGAs are also success in many problems despite they have higher-order alphabet and they successfully *hill-climb* on the search space. Since [8] showed that Holland's Schema theorem holds for some sort of crossing-over operators, FPGAs can still be considered as genetic algorithms although they have *phenotype* operators. A well-described research history on floating-point evolutionary algorithms is given in [7].

Since FPGAs are directly performed on the *phenotype*, the bitwise crossing-over and mutation operators are not applicable. As a result of this, many classes of these operators are developed. *Flat Crossover* and *Simple Crossover* mimic the k-point crossing-over operator in GAs by combining parameters corresponding to the same locations of parents [9, 10]. [11] introduced the *Simulated Binary Crossover (SBX)* which has a search power similar to *one-point crossover* of binary GAs. [12] devised the *Blend Crossover (BLX-α)* which is based on linear combinations of parents and product of their differences and a constant α. They have reported that the special case of $\alpha = 0.5$ outperforms all other scenarios that are performed with different values of α. [13] suggested *BLX-α* to apply randomly selected genes with a given probability rather than the whole chromosome and they stated that the generated offspring will still hold the range constraints unless $\alpha > 1$. They also suggested in their book to select the α randomly

within the range $[0,1]$. In our simulations, we follow the instructions given in [13] for crossing-over operator. A comparative description of some other crossing-over operators can be found in [14].

In GAs, the mutation operator simply flips a randomly selected bit of a chromosome, however, this definition of perturbation does not easily take a place in FPGAs. Several mutation operators are developed. [10] introduced the *random* mutation operator which randomly changes a parameter in the given range. The *non-random* mutation operator modifies a parameter using a function of current number of generation and maximum number of generations [10]. In general, these operators change some parameters *randomly*, *as a function of generations* or both. *Random* mutation can be applied by changing a parameter's value using a random variable that follows a *Uniform(a,b)* distribution as well as a *Normal(μ, σ^2)* distribution. While the parameters a and b of *Uniform* distribution are naturally defined by parameter bounds, it is easy to control. However, selecting the right σ^2 may require a second stage optimization of FPGA parameters [13]. A comparative description of some other mutation operators can be found in [14]. In our simulation, as the mutation operator, we simply select a random number from a *Uniform* distribution with parameters of corresponding bounds.

3. Proposed operators

[8] stated that the mutation operator in FPGAs alters parameters individually and changing a parameter's value may increase the fitness, whereas, changing an other one may create an opposite effect. As a solution, they suggest changing all of the parameters included in a chromosome by mutation, however, it would not be compatible with Schemata theorem.

Suppose that a floating-point chromosome C contains m parameters and takes values $c_1, c_2, ..., c_m$. The optimum and unknown solution vector S has values $s_1, s_2, ..., s_m$. Now we define a direction vector D which contains $+1$s, -1s and 0s as its elements at position d_i. If d_i is $+1$, a positive mutation change is required to achieve s_i, whereas, if d_i is -1, c_i should be mutated in opposite direction. Similarly, if d_i is 0, c_i should not be changed. Assuming that s_i and s_j are independent for all i and j except $i \neq j$, the probability of performing mutation operator on all parameters in right directions is $P_{best} = \binom{m}{m}(\frac{1}{3})^m(1-\frac{1}{3})^{m-m} = (\frac{1}{3})^m$. On the contrary, after performing a

mutation with a predefined probability $P_{mutation}$, probability of having at least one parameter changed in right direction (or not being changed) is $P_a = 1 - \binom{m}{0}(\frac{1}{3})^0(1-\frac{1}{3})^{m-0} = 1 - (\frac{2}{3})^m$ which almost always implies that $P_{best} < P_a$ for $m > 1$. As a result of this, a mutation operator that is capable to modify all parameters in right directions should improve the performance and convergency.

Hooke-Jeeves (HJ) algorithm [15, 16] is a local search algorithm which can be applied in optimization problems that may have a non-differentiable goal function. Suppose that the goal function under consideration is $\min / \max f(x)$ where x is an m dimensional parameter vector. The algorithm starts with an initial base point x_i. The base point is then moved in all m dimensions and successful moves are combined for later moves. If the new base point is better than the old one, initial base point is set to new one. Algorithm adjusts the step length at each iteration. If the step length gets smaller than the minimum step length, algorithm stops.

The study reported in [17] combines evolution strategies (ES) as a first stage global optimization and HJ as a second stage local optimization. In this study, ES is performed to find good solutions near the global optimum and HJ is applied for local fine-tuning.

Hybridization a global search algorithm with a local search algorithm as an *intelligent mutation* is not new [18]. [19] combined evolutionary algorithms for travelling salesman problem to improve the candidate solutions in the population. [20] suggested to combine GA with Simulated Annealing (SA) in a way that SA based operators may replace or co-operate with the standard GA operators.

Now suppose *HJOP* is a genetic operator defined as

$$HJOP(x_i, fn, x_{min}, x_{max}), \tag{1}$$

where x_i is base point for HJ algorithm, fn is the evolution function, x_{min} is an m-vector of lower bounds of parameters, x_{max} is an m-vector of upper bounds of parameters, m is the chromosome length or number of parameters. The operator performs a local search and generates a better solution with a predefined probability P_{HJOP}. In our simulations, only the best solution of the current population is selected as base point and both the parent and the generated offspring are hold in the next generation. A pseudo-code for FPGA with HJOP is defined in Table 1.

```
pop <- generateInitialPopulation()
temporary.pop <- generateEmptyPopulation()
while(currentIteration < maxNumberOfIterations){
    calculateFitness()
    applyElitism()
    if(probability(HJOP) < random) {
       applyHJOP()
       copyOffSpringTo(temporary.pop)
    }
    foreach (chromosomes in pop){
       selectParents()
       if(probability(crossover) < random) applyCrossOver()
       if(probability(mutation) < random) applyMutation()
       copyOffSpringTo(temporary.pop)
    }
    swap(pop, temporary.pop)
}
```

TABLE 1. FPGA with HJOP

4. Simulations

We perform a simulation study to measure and test the effects of proposed operator. A set of test functions used in simulations with several dimensions $p = 5, 25, 50$ and $p = 100$. This set of functions includes *Ackley*, *Bohachevsky*, *Griewank*, *Rastrigin*, *Scaled Rastrigin* and *Skew Rastrigin* which are previously used in comparison of several optimization methods [21, 22, 6].

Simulations are performed in *R* software [23]. Each single function in the test set is used as cost function to minimize for several number of parameters. Simulations are run for the constant population size (50), crossover probability (0.80) and mutation probability (0.05). In each single generation, the best solution is directly copied to next one. Standard *FPGA* and *FPGA* with

HJOP have been initiated using same randomized populations. Simulations are performed 100 times for each configuration. *HJOP* is applied with the probability of $P_{HJOP} = 0.10$, so the expected number of runs is 5 for the selected population size. Maximum number of generations is set to 100 as the stopping criterion. Table 2-3 summarize the results.

In Table 2-3 minimum, median, maximum and mad (median absolute deviations) of final values of cost functions are reported for all functions and cases. The column *#success* shows the number of cases in which the final cost value is smaller than $\varepsilon = 1/100000$. The column $p-value$ represents the P-values calculated after performing Mann-Whitney test of equality of location parameters. The *null*-hypothesis of this test is $H_0 : \mu_1 = \mu_2$, where μ_1 and μ_2 are the location parameters of final cost values resulted from *FPGA* runs with and without the *HJOP*, respectively. The tests are performed with the alternative hypothesis $H_a : \mu_1 \neq \mu_2$. First row of Table 1 represents the results of Ackley function with 5 parameters resulted by standard FPGA, whereas, the following line holds the statistics of results obtained by FPGA with *HJOP*. It can be easily seen that *HJOP* significantly increase the performance when the cost function is Ackley with parameters $5, 25, 50$ and 100. As it can be seen in Table 2, the difference between the global minimum and the final result obtained by FPGA with *HJOP* is smaller than ε in all iterations when the cost function is Ackley. In Table 3, it is shown that the results are similar for functions *Bohachevsky*, *Rastrigin* and *Skew Rastrigin*. Number of finding success solutions decreases for functions *Griewank*, *Rastrigin* and *Scaled Rastrigin*. However, *min*, *median* and *max* statistics are drastically reduced by using *HJOP*. *P-values* are considerable 0 and the null hypothesis can be rejected for all cases.

Addition to simulations, we perform the classical FPGA and FPGA with *HJOP* for all functions with 25 parameters and the effect of the operator is shown in Figure 1. The search history of FPGA runs is shown by solid lines[1], whereas, dashed lines shows the search history of FPGA with *HJOP* by iterations. It is shown that applying the operator in a single generation directly jumps the best chromosome onto the global optimum for Ackley. While *Bohachevsky*, *Rastrigin* and *Skew Rastrigin* need the operator applied for two times to be minimized, the operator

[1]Logarithm of cost values by iterations

is applied many times for functions *Griewank* and *Scaled Rastrigin* for reaching the global optimum after many steps. The graphics in Figure 1 also show the effects of hybridization as an intelligent mutation operator applied using a predefined probability.

5. Conclusion

Hybridization of GAs is handled in many ways in GA literature. The most common hybridization technique is the two stage optimization in which the second stage is consist on performing a local search algorithm after a genetic search. GAs are successful global optimization techniques if the initial population is well-randomized over the search space. Addition to this, the global optimum may not be reached using the standard genetic operators and a fine-tuning operation may be needed to mutate best solution. If the diversity of population is not provided and/or the genetic operators do not modify the parameters in the correct directions, the algorithm is said to be got stuck on local optimum and/or reached a solution which is not optimum, that is, performing a local-search algorithm may not help to find the global optimum by searching wrong subspaces.

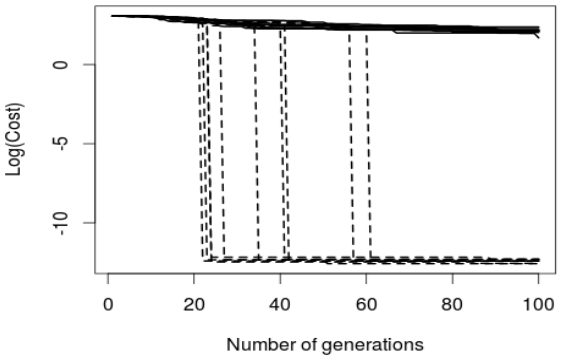
In this paper, it is shown that applying a local optimizer that performs a search in all dimensions as a genetic operator is better than the standard mutation operator. We suggest to apply Hooke-Jeeves algorithm on the best candidate solution as the base point with a given probability. This use of algorithm can be considered as a genetic operator in FPGAs. We perform a simulation study on some well-known test functions to measure the effect of proposed algorithm. Simulation results show that the proposed operator has a significant effect on reaching global optimum even a limited configuration of FPGA.

	p	min	$median$	max	mad	$\#success$	$p-value$
Ackley	5	0.001	0.217	2.877	0.281	0	
		0.000	0.000	0.000	0.000	100	0.000
	25	6.633	9.092	11.832	1.408	0	
		0.000	0.000	0.000	0.000	100	0.000
	50	10.471	14.577	17.467	1.867	0	
		0.000	0.000	0.000	0.000	100	0.000
	100	14.677	18.346	19.629	0.821	0	
		0.000	0.000	0.000	0.000	100	0.000
Bohachevsky	5	0.000	0.056	4.851	0.081	2	
		0.000	0.000	0.000	0.000	100	0.000
	25	143.646	342.090	863.798	104.074	0	
		0.000	0.000	237.851	0.000	98	0.000
	50	1380.729	3480.890	9140.196	1344.657	0	
		0.000	0.000	0.000	0.000	100	0.000
	100	6193.256	21980.925	64276.650	6556.754	0	
		0.000	0.000	0.000	0.000	100	0.000
Griewank	5	0.000	0.042	0.928	0.061	1	
		0.000	0.010	0.713	0.015	41	0.000
	25	0.974	1.036	1.092	0.014	0	
		0.000	0.000	1.139	0.000	62	0.000
	50	1.107	1.290	1.600	0.105	0	
		0.000	0.000	1.694	0.000	70	0.000
	100	1.620	2.806	5.837	0.584	0	
		0.000	0.000	1.712	0.000	72	0.000

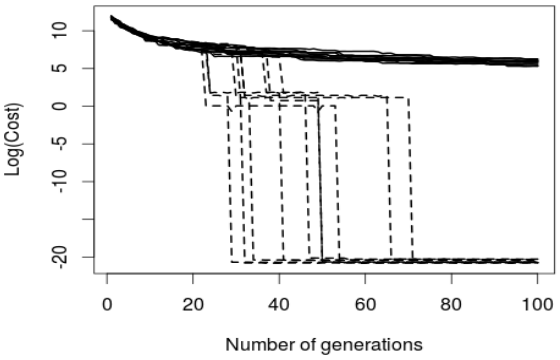
TABLE 2. Simulation results for test functions

	p	*min*	*median*	*max*	*mad*	*#success*	$p-value$
Rastrigin	5	0.001	14.666	50.936	12.232	0	
		0.000	0.000	0.000	0.000	100	0.000
	25	3747.692	11387.300	30024.090	4886.346	0	
		0.000	0.000	0.000	0.000	100	0.000
	50	51863.640	122888.750	243857.800	42201.053	0	
		0.000	0.000	0.000	0.000	100	0.000
	100	232337.600	689377.300	1397417.000	246576.024	0	
		0.000	0.000	0.000	0.000	100	0.000
Scaled	5	5.586	33.750	151.054	21.784	0	
Rastrigin		0.000	15.504	60.273	11.257	2	0.000
	25	23206.830	77407.860	341140.500	42706.723	0	
		8.165	158.434	492.363	74.097	0	0.000
	50	204635.600	610189.150	2385967.000	275840.250	0	
		15.715	288.188	548.349	155.560	0	0.000
	100	1031361.000	2585665.000	5162986.000	884797.889	0	
		42.300	557.990	1234.115	272.498	0	0.000
Skew	5	345.347	1687.564	6429.002	990.974	0	
Rastrigin		0.000	0.000	0.000	0.000	100	0.000
	25	249186.000	567986.300	932628.200	108116.826	0	
		0.000	0.000	0.000	0.000	100	0.000
	50	774956.400	1431846.000	2390903.000	282037.963	0	
		0.000	0.000	0.000	0.000	100	0.000
	100	1580999.000	4031329.000	10747192.000	1317314.563	0	
		0.000	0.000	0.000	0.000	100	0.000

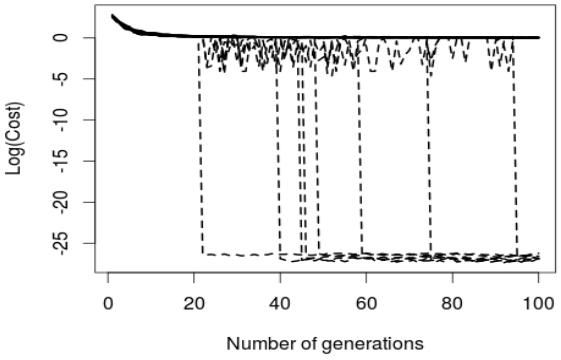
TABLE 3. Simulation results for test functions



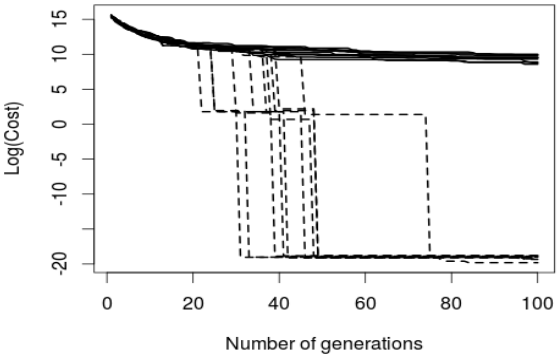
(A) Ackley



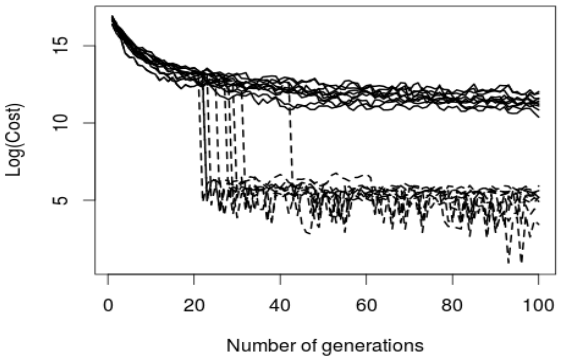
(B) Bohachevsky



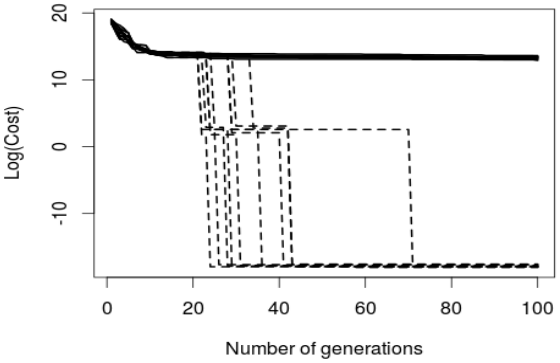
(C) Griewank



(D) Rastrigin



(E) Scaled Rastrigin



(F) Skew Rastrigin

FIGURE 1. Effect of local search operator

Conflict of Interests

The author declares that there is no conflict of interests.

REFERENCES

[1] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence, U Michigan Press, 1975.

[2] D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1989.

[3] M. Srinivas and L. M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, Systems, Man and Cybernetics, IEEE Transactions, 24 (1994), 656–667.

[4] R. K. Ursem, Diversity-guided evolutionary algorithms, Parallel Problem Solving from NaturePPSN VII, Sringer, 2002.

[5] S.-F. Hwang and R.-S. He, A hybrid real-parameter genetic algorithm for function optimization, Adv. Eng. Informatics, 20 (2006), 7–21.

[6] M. H. Satman, Machine coded genetic algorithms for real parameter optimization problems, Gazi Univ. J. Sci. 26 (2013), 85–95.

[7] D. E. Goldberg, Real-coded genetic algorithms, virtual alphabets, and blocking, Urbana, 1990.

[8] A. H. Wright, et al., Genetic algorithms for real parameter optimization, FOGA, 1990.

[9] N. J. Radcliffe, Equivalence class analysis of genetic algorithms, Complex systems, 5 (1991), 183–205.

[10] Z. Michalewicz, Genetic algorithms+ data structures= evolution programs, springer, 1996.

[11] R. B. Agrawal and K. Deb, Simulated binary crossover for continuous search space, tech. rep., 1994.

[12] L. J. Eshelman, chapter real-coded genetic algorithms and interval-schemata, Foundations of genetic algorithms, 2 (1993), 187–202.

[13] R. L. Haupt and S. E. Haupt, Practical genetic algorithms, John Wiley & Sons, 2004.

[14] F. Herrera, M. Lozano, and J. L. Verdegay, Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis, Artificial Intelligence Review, 12 (1998), 265–319.

[15] R. Hooke and T. A. Jeeves, Direct search solution of numerical and statistical problems, J. Appl. Comput. Math. 8 (1961), 212–229.

[16] I. Moser, Hooke-jeeves revisited, Evolutionary Computation, 2009. CEC'09. IEEE Congress on, pp. 2670–2676, IEEE, 2009.

[17] R. O. Bowden and J. D. Hall, Simulation optimization research and development, Simulation Conference Proceedings, 1998. Winter, vol. 2, pp. 1693–1698, IEEE, 1998.

[18] T. Bäck, D. B. Fogel, and Z. Michalewicz, Evolutionary computation 1: Basic algorithms and operators, vol. 1. CRC Press, 2000.

[19] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, Evolution algorithms in combinatorial optimization", Parallel Computing, 7 (1998), 65–85.

[20] D. Adler, Genetic algorithms and simulated annealing: A marriage proposal, Neural Networks, 1993., IEEE International Conference, pp. 1104–1109, IEEE, 1993.

[21] E. P. Adorio and U. Diliman, Mvf-multivariate test functions library in c for unconstrained global optimization, 2005.

[22] N. Hansen and S. Kern, Evaluating the cma evolution strategy on multimodal test functions", Parallel Problem Solving from Nature-PPSN VIII, pp. 282–291, Springer, 2004.

[23] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2014.