Comparison of a Crossover Operator in Binary-coded Genetic Algorithms

STJEPAN PICEK
Faculty of Electrical Engineering
and Computing
Unska 3, 10000 Zagreb
CROATIA
stjepan@computer.org

MARIN GOLUB
Faculty of Electrical Engineering
and Computing
Unska 3, 10000 Zagreb
CROATIA
marin.golub@fer.hr

Abstract: Genetic algorithms (GAs) represent a method that mimics the process of natural evolution in effort to find good solutions. In that process, crossover operator plays an important role. To comprehend the genetic algorithms as a whole, it is necessary to understand the role of a crossover operator. Today, there are a number of different crossover operators that can be used in binary-coded GAs. How to decide what operator to use when solving a problem? When dealing with different classes of problems, crossover operators will show various levels of efficiency in solving those problems. A number of test functions with various levels of difficulty has been selected as a test polygon for determine the performance of crossover operators. The aim of this paper is to present a larger set of crossover operators used in genetic algorithms with binary representation and to draw some conclusions about their efficiency. Results presented here confirm the high-efficiency of uniform crossover and two-point crossover, but also show some interesting comparisons among others, less used crossover operators.

Key–Words: Evolutionary computation, Genetic algorithms, Crossover operator, Efficiency, Binary representation, Test functions

1 Introduction

Genetic algorithms (GAs) represent powerful general purpose search method based on evolutionary ideas of natural selection and genetics. They simulate natural processes based on principles of Lamarck and Darwin. Genetic algorithms were invented by John Holland and later developed by Holland and his students and colleagues in the 1960s and 1970s. Contrary to evolution strategies or evolutionary programming, Holland's original goal was to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. Holland's introduction of a population based algorithm with operators crossover, inversion, and mutation was a important innovation. Today, there exists many variations on genetic algorithms and term ”genetic algorithm” is used to describe concepts sometimes very far from Holland's original idea [7]. The two most commonly employed genetic search operators are crossover and mutation. Crossover produces offspring by recombining the information from two parents. It is the major exploratory mechanism of the genetic algorithms. Mutation prevents convergence of the population by flipping a small number of randomly selected bits to continuously introduce variation. The driving force behind genetic algorithms is the unique cooperation between selection, crossover and mutation operator. [9]
The main goal of this article is to parallely present the performance of crossover operators that are not in wide use. Results present in current literature usually display the performance of one less commonly used operator to the other, more widely used crossover operator like uniform or single-point crossover. In section 2 the necessary background informations are presented, section 3 defines the parameters used in the experiments, results obtained from the experiments and presents a discussion about results, and finally, section 4 draws a conclusion.

2 Background

2.1 Test Functions

Test functions used in this article have two important features: modality and separability.
Unimodal function is a function with only one global optimum.
Function is multimodal if it has two or more local optima. Multimodal functions are more difficult to optimize compared to unimodal functions.
Function with *n* variables is separable if it can be written as *n* functions of one variable e.g. it can be broken down into functions with fewer independent variables. The notion of separability is related to the epistasis, where epistasis represent coupling between different parameters of a cost function.
Functions that cannot be separated are more difficult to solve as the correct search direction depends on two or more genes.
All functions used for experiments are functions of one variable. [16]

2.1.1 Sphere Function - F1.

Sphere function is a test function proposed by De Jong. It has been widely used in evaluation of genetic algorithms and development of the theory of evolutionary strategies. Sphere function is a simple, continuous and strongly convex function. Sphere function is unimodal and additively separable. Boundaries are set at $[-5.12, 5.12]$. Sphere function's global minimum is in point *x*=0 with value *f(x)*=0. [16][17]

$$f(x) = \sum_{i=1}^{D} x_i^2 \, . \tag{1}$$

2.1.2 Axis Parallel Hyper-Ellipsoid Function - F2.

This function is similar to Sphere function. It is also known as weighted sphere model. It is also unimodal and additively separable. Boundaries are set at $[-5.12, 5.12]$. Function's global minimum is in point *x*=0 with value *f(x)*=0. [16][17]

$$f(x) = \sum_{i=1}^{D} i * x_i^2 \, . \tag{2}$$

2.1.3 Rotated Hyper-Ellipsoid Function - F3.

This function represents an extension of the axis parallel hyper-ellipsoid function. With respect to the coordinate axes, this function produces rotated hyper-ellipsoids. It is continuos, convex and unimodal. Boundaries are set at $[-65.536, 65.536]$. Function's global minimum is in point *x*=0 with value *f(x)*=0. [16][17]

$$f(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2 \, . \tag{3}$$

2.1.4 Normalized Schwefel Function - F4.

The surface of Schwefel function is composed of a great number of peaks and valleys. The function has a second best minimum far from the global minimum where many search algorithms are trapped. Moreover, the global minimum is near the bounds of the domain. Schwefel's function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minimum. Schwefel function is multimodal and additively separable. Boundaries are set at $[-500, 500]$. Function's global minimum is in point *x*=420.968 with value *f(x)*=-418.9829. [16][17]

$$f(x) = \sum_{i=1}^{D} -x_i * \sin\left(\sqrt{|x_i|}\right) \, . \tag{4}$$

2.1.5 Generalized Rastrigin Function - F5.

Rastrigin function was constructed from Sphere adding a cosine modular term. Its contour is made up of a large number of local minima whose value increases with the distance to the global minimum. Thus, the test function is highly multimodal. However, the location of the local minima's are regularly distributed. Rastrigin function is additively separable. Boundaries are set at $[-5.12, 5.12]$. Function's global minimum is found in point *x*=0 with value *f(x)*=0. [15][16][17]

$$f(x) = 10 * n + \sum_{i=1}^{D} \left(x_i^2 - 10 * \cos(2 * \Pi * x_i) \right) \, . \tag{5}$$

2.1.6 Salomon Function - F6.

Salomon function is multimodal, additively separable function. Boundaries are set at $[-5.12, 5.12]$. Function's global minimum is found in point *x*=0 with value *f(x)*=0. [16][17]

$$f(x) = -1 * \cos\left(2 * \Pi * \sqrt{\sum_{i=1}^{D} x_i^2}\right) + 0.1 * \sqrt{\sum_{i=1}^{D} x_i^2} + 1 \ . \quad (6)$$

2.1.7 Ackley's Path Function - F7.

Ackley Path function, originally proposed by Ackley and generalized by Baeck has an exponential term that produces numerous local minima. The complexity of this function is moderated. An algorithm that only uses the gradient steepest descent will be trapped in local optima, but any search strategy that analyzes a wider region will be able to cross the valley among the optima and achieve better results. In order to obtain good results for this function, the search strategy must combine the exploratory and exploitative components efficiently. This function is multimodal and not separable. Boundaries are set at $[-32.768, 32.768]$. Function's global minimum is found in point *x*=0 with value *f(x)*=0. [16][17]

$$f(x) = -20 * \exp^{-0.2\sqrt{\frac{\sum_{i=1}^{D} x_i^2}{D}}} - N \quad (7)$$

where *N* equals

$$N = \exp^{\frac{\sum_{i=1}^{D} \cos(2*\Pi*x_i)}{D}} + 20 + \exp^1 \ . \quad (8)$$

2.1.8 Michalewicz Function - F8.

The Michalewicz function is a multimodal test function with $n!$ local optima. The parameter m defines the "steepness" of the valleys or edges. Larger *m* leads to more difficult search. For very large *m* the function behaves like a needle in the haystack - the function values for points in the space outside the narrow peaks give very little information on the location of the global optimum. In experiments the size of parameter *m* was 10. Boundaries are set at $[0, \Pi]$. Optimal fitness value for this function has value *f(x)*=-9.66. [16][17]

$$f(x) = -1 * \sum_{i=1}^{D} \sin(x_i) \left(\sin\left(\frac{i * x_i^2}{\Pi}\right)\right)^{2*m} \ . \quad (9)$$

2.1.9 Griewangk Function - F9.

Griewangk's function is similar to Rastrigin's function. It has many widespread local minima. However, the location of the local minima's are regularly distributed. Boundaries are set at $[-600, 600]$. Function's global minimum is found in point *x*=0 with value *f(x)*=1. [16][17]

$$f(x) = -\sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \ . \quad (10)$$

2.2 Position-Dependent Bias

In binary-coded GAs possible solutions are represented as strings of bits called chromosomes. Crossover operator will more likely disrupt bits that are relatively far apart. Position-dependent bias states that interacting bits which are relatively far apart on string are more likely to be separated than bits that are relatively close together. Analogous to this, non-interacting bits that are close together will more likely be preserved than bits which are far apart. [2][9]

2.3 Deception

Based on the works of Bethke (1980) and Goldberg (1987), the term 'deception' has been introduced in order to better understand the situations when genetic algorithm might fail. These deceptive functions create a problem for the genetic algorithm when performing optimization tasks. Deceptive functions are a family of fitness landscapes developed to challenge the building block hypothesis. The building block hypothesis is a direct result of schema theorem and it states that genetic algorithms work by combining low-order building blocks to form higher order ones. The problem is fully deceptive when all the relevant lower order hyperplane competitions lead toward a deceptive attractor.[10][11] When talking about deceptive functions, Whitley stated
"Deception implies that the global winner of some hyperplane competition of order-N has a bit pattern that is different from the bit pattern of the 'global winner' for some 'relevant' lower order competition." [14]
One example of a deceptive function is Schwefel function.

2.4 Various Crossover Operators

Amongst all evolutionary algorithms, genetic algorithms have a recombination operation which is probably closest to the natural paragon. The crossover operator is used to mimic biological recombination between two single chromosome organisms. Rough

definition of crossover can be that it represents variously chosen rules of exchange. In his work, Holland also used mutation, but in that scheme it is generally treated as subordinate to crossover. Thus, in Holland's GA, instead of the search moving from point to point as in neighborhood search approaches, the whole set of strings undergoes 'reproduction' in order to generate a new population. For fixed-length strings, the crossover points for both parents are always identical. [3][5]
For purpose of this work, only crossover operators that operate on two parents and have no self-adaptation properties will be considered. In first set of experiments there is no crossover operator - C0.

2.4.1 Single-Point Crossover - C1.

When performing crossover, both parental chromosomes are split at a randomly determined crossover point. Subsequently, a new child genotype is created by appending the first part of the first parent with the second part of the second parent. [3][5]

2.4.2 Two-Point Crossover - C2.

In two-point crossover, both parental genotypes are split at two points, constructing a new offspring by using parts number one and three from the first, and the middle part from the second ancestor. When using two-point crossover we can expect poorer performance results because building blocks are more likely to be disrupted. From other point of view using two-point crossover will enable searching problem space more thoroughly. Using single-point and two-point crossover operator prevents schema to be disrupted, but when population becomes homogeneous, search space becomes smaller. [3][5]

2.4.3 Uniform Crossover - C3.

Single and multi-point crossover defines cross points as places between loci where an individual can be split. Uniform crossover generalizes this scheme to make every locus a potential crossover point. A crossover mask, the same length as the individual structure is created at random and the parity of the bits in the mask indicate which parent will supply the offspring with which bits. To avoid problems with genes locus, it is good to use uniform crossover. Uniform crossover disrupts schema with great probability but searches larger problem space. For uniform crossover, the number of effective crossing points is not fixed, but will average to $l/2$ where l represents string length. [3][5][6][17]

2.4.4 Half-Uniform Crossover - C4.

Half-Uniform crossover is similar to uniform crossover. Only difference is that only half of differing bits between parents will be swapped. [18]

2.4.5 Reduced Surrogate Crossover - C5.

To reduce the chance of producing clones Booker suggested examining the selected parents to define suitable crossover points. A reduced surrogate crossover operator reduces parent strings to a skeletal form in which only those bits that differ in two parents are represented. Recombination is then limited only to positions of bits in reduced surrogates. Single-point crossover was used for recombination of skeletal forms of parents. Single-point crossover operator can produce parents' clones; to avoid that reduced surrogate crossover should be used. If at least one crossover point occurs between the first and last bits in reduced surrogate, then the offspring will never duplicate the parents. Also, reduced surrogate will cause that recombination process equally weight the probability of generating each offspring which can potentially be produced by an operator. Single-point crossover in any continuous region of matching bits in parents produces same offspring, and thus introducing bias for some offspring. Reduced surrogate removes that kind of potential bias. [1][3][17]

2.4.6 Shuffle Crossover - C6.

Shuffle crossover is similar to one-point crossover. First, a single crossover position is selected. Before the variables are exchanged, they are randomly shuffled in both parents. After recombination, the variables in the offspring are unshuffled in reverse. This removes positional bias as the variables are randomly reassigned each time crossover is performed. In a way, shuffle crossover is similar to uniform crossover. Difference is that uniform crossover exchanges bits and not segments like shuffle crossover. Further, in uniform crossover bits exchanged follow a binary distribution and in shuffle crossover bits follow uniform distribution, as in single-point crossover. [2][3]

2.4.7 Segmented Crossover - C7.

Segmented crossover represent a variant of N-point crossover. In this crossover the number of crossover points is not constant. Fixed number of crossover points is replaced by segment switch rate *s*, which specifies the probability that segment will end at any point in the string. Starting from first position in a string, one real-valued number *q* and one natural number *j* are generated. The number *q* represents the probability that *j* will be crossover point. In experiments value 0.2 was used as a segment switch rate *s*. [3][8]

3 Main Results

In all experiments generational binary-coded GA with binary tournament selection was used. Parameters of genetic algorithm for the first two rounds of experiments were as following: bit-wise mutation with p_m mutation coefficient of 0.01 and crossover rate p_c of 0.8 were used, number of independent runs for each experiment was 30, initial population *N* of size 30, 50 and 100 was randomly created and used in experiments. String length *l* of 15 bits was used. Every solution in search space has the same precision but for the functions with smaller domain size the scale for possible solutions is larger. Dimensionality of the search space *D* for all test function was set to 30. Number of overall evaluations *H* was set to 10000. Number of generations was obtained by dividing number of overall evalations *H* with population size *N*. For all test functions finding global minimum is the objective.
For purpose of this article, three sets of experiments were conducted. In first set, overall perfomance of all crossover operators on all test functions were evaluated. In second set, two test functions that displayed the most interesting behaviour were tested more rigorously. Finally, in third set, two crossover operators that displayed best perfomance in second set were tested with various values of crossover coefficient. In first set of experiments all crossover operators was tested against all test functions. Test case when no crossover operator was used, e.g. operator C0 has continuously showed the worst performance. That kind of result was somewhat expected because for many problems, especially more difficult ones, only mutation operator don't have enough exploration strength to reach to the optimum. The successfulness of C0 operator was largely due to distribution of solutions in initial population.
Other crossover operators showed similar results, at least in reaching optimum, for test functions F1, F2, F3, F6, F7 and F9. Single-point crossover performed poorer than two-point crossover in every test case. Uniform crossover outperformed half-uniform crossover for all test cases except F8. Reduced surrogate crossover displayed poorer results when comparing to all, except to single-point crossover. Segmented crossover had slightly better results than shuffle crossover. Uniform crossover produces best results in majority of the cases, however, there are problems, like Michalewicz function where two-point crossover outperforms uniform crossover. When evaluating the speed of a convergence, where convergence represents evolving of the solutions towards global optimum, best results had achieved uniform and two-point crossover. In all of the test functions shuffle crossover showed the slowest convergence. GA that used shuffle crossover succeeded in reaching the optimum, but the average value of the solutions was by far the worst of all tested crossover operators. In table 1. value INF represents any number too big to be represented by floating point number type in C programming language.
Test functions F4 and F8 (Schwefel and Michalewicz function) were then selected for more in-depth analysis.
In second round of the experiments the accent was set to the varying size of the initial population, so three population sizes were used: small of 30 individuals, medium of 50 individuals and large of 100 individuals.
Although smaller populations had more generations, results obtained were not as good as for larger populations. A conclusion can be developed that large populations make better results for test functions that were used. [1][3]
In third set of experiments tests were conducted for 5 different p_c coefficients and most successful crossover operators from second set of experiments - C2 and C3. Better results were achieved for larger p_c, especially for coefficient of 0.9. Conclusion can be made that for difficult problems, by varying the crossover coefficient some improvement can by observed, but in general, that would not be enough to reach global optimum. Rather, some other technique that utilize more exploitation properties should be used at that moment.

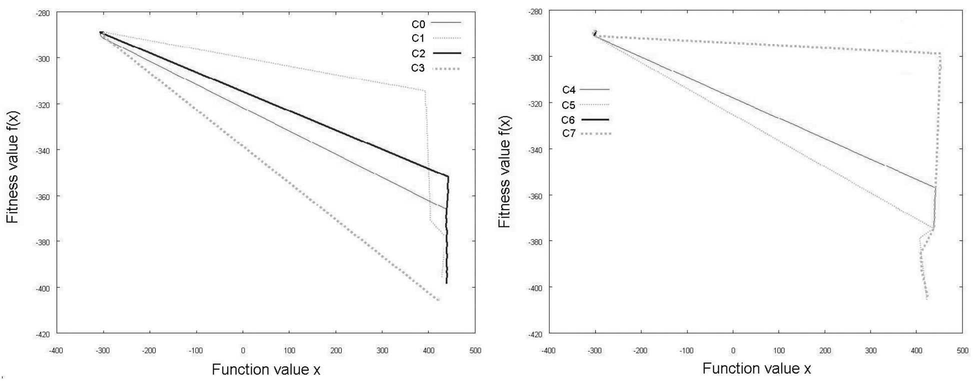

Figure 1: (a) Fitness values for crossover operators 0 to 3 and Schwefel function, (b) Fitness values for crossover operators 4 to 7 and Schwefel function

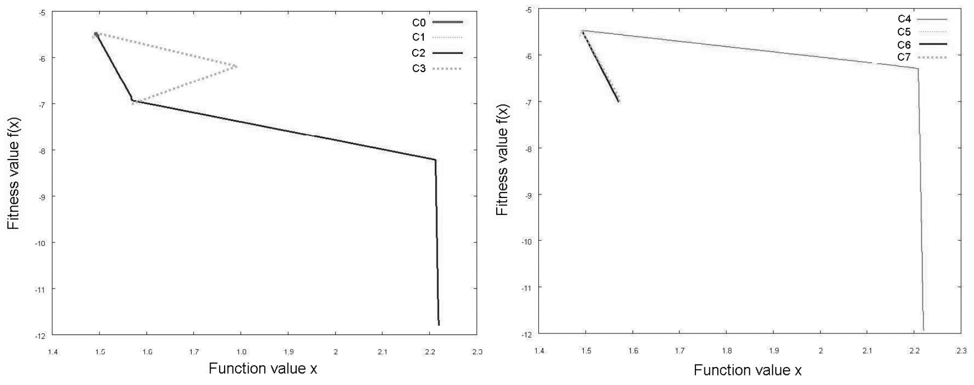

Figure 2: (a) Fitness values for crossover operators 0 to 3 and Michalewicz function, (b) Fitness values for crossover operators 4 to 7 and Michalewicz function

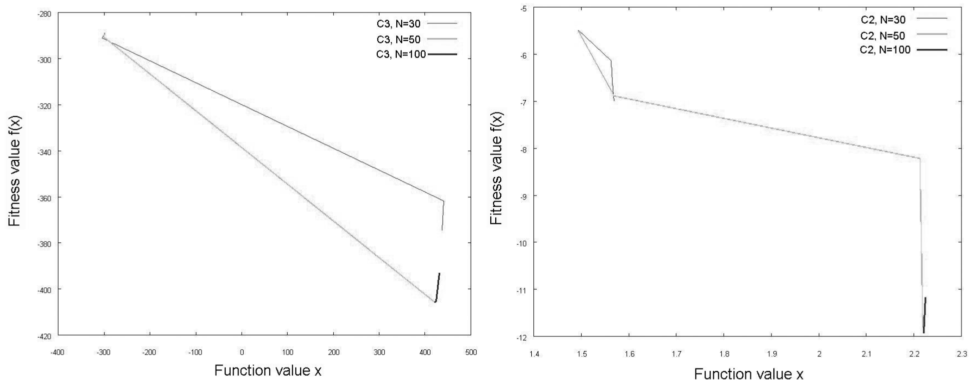

Figure 3: (a) Fitness values for Schwefel function with crossover operators C3 and various population sizes, (b) Fitness values for Michalewicz function with crossover operators C2 and various population sizes

Table 1: Results for experiments with N=50 population size

Func.	Result	C0	C1	C2	C3	C4	C5	C6	C7
F1	best	0.25	1E-7	1E-7	1E-7	1E-7	1E-7	1E-7	1E-7
	worst	760.21	760.21	760.21	760.21	760.21	760.21	760.21	760.21
	average	29.34	28.73	29.25	15.03	32.89	30.21	361.12	19.52
F2	best	3.86	1.1E-5	1.1E-5	1.06E-5	1.06E-5	1.1E-5	1.1E-5	1.1E-5
	worst	712.9	2.86E3	735.3	2.86E3	2.85E3	3.61E3	1.14E4	2.41E3
	average	21.42	56.14	15.98	56.14	55.70	71.02	6154.6	484.31
F3	best	INF	3.35E-2	3.35E-2	2.87E-2	3.35E-2	2.87E-2	3.35E-2	3.35E-2
	worst	INF	INF	INF	INF	INF	INF	INF	INF
	average	INF	INF	INF	INF	INF	INF	INF	INF
F4	best	-374.71	-395.97	-397.02	-405.7	-374.71	-405.7	-291.19	-404.19
	worst	280.42	288.94	286.18	286.3	287.15	284.5	374.3	286.23
	average	-350.05	-378.61	-374.71	-392.15	-390.97	-393.36	192.02	-353.34
F5	best	45.88	38.85	10.0	10.0	10.0	38.85	10.0	10.0
	worst	646.97	939.37	732.77	496.49	608.65	481.18	879.46	848.81
	average	81.89	112.96	37.81	19.36	31.66	39.52	471.42	33.95
F6	best	0.73	0.20	0.00	0.00	0.00	0.02	0.00	0.31
	worst	4.19	4.66	4.65	4.19	4.19	4.19	4.74	4.41
	average	2.35	2.19	2.35	1.09	2.290	2.35	2.81	1.79
F7	best	3.382	0.58	0.28	0.09	0.09	0.41	0.09	0.09
	worst	22.21	21.45	13.72	21.43	21.57	22.14	22.15	20.05
	average	17.82	5.99	4.28	4.22	8.79	5.18	13.73	3.89
F8	best	-5.48	-5.59	-11.81	-7.01	-11.93	-5.59	-7.01	-7.01
	worst	-0.09	-0.52	-0.72	0.00	-0.96	-0.52	0.00	0.00
	average	-4.45	-5.47	-11.11	-6.68	-11.79	-5.47	-3.04	-6.72
F9	best	1.85	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	worst	2.43E3	2.1E3	2.52E3	1.8E3	2.48E3	1.72E3	2.52E3	2.12E3
	average	791.89	363.99	413.84	524.65	791.89	378.6	980.1	414.36

Table 2: Results for functions F4 and F8 and various population sizes

Func.	P. size	Result	C0	C1	C2	C3	C4	C5	C6	C7
F4	30	best	-337.7	-374.7	-405.7	374.7	-374.7	-374.7	-291.2	-374.7
		worst	405.2	275.2	290.9	275.2	280.4	275.3	374.3	271.9
		average	-241.64	-353.13	-343.50	-353.15	-347.94	-353.13	-162.79	-360.91
	50	best	-374.7	-395.9	-397.02	-405.70	-374.71	-405.70	-291.19	-404.19
		worst	280.42	288.94	286.18	286.30	287.15	284.51	374.30	286.24
		average	-350.05	-378.61	-374.71	-392.15	-390.97	-393.36	192.02	-353.34
	100	best	-393.17	-405.85	-405.97	-405.97	-405.97	-405.97	-405.64	-405.70
		worst	397.69	286.18	288.94	42.14	288.94	42.14	284.63	287.62
		average	-347.39	-384.82	-371.70	-386.69	-375.19	-376.69	-223.58	-377.97
F8	30	best	-5.46	-7.01	-7.00	-7.01	-5.59	-7.01	-7.01	-5.59
		worst	0.00	-5.15	0.00	-5.16	-5.03	-5.16	0.00	-4.92
		average	-3.44	-6.89	-6.65	-6.89	-5.56	-6.89	-2.72	-5.55
	50	best	-5.48	-5.59	-11.81	-7.01	-11.93	-5.59	-7.01	-7.01
		worst	-0.09	-0.52	-0.71	0.00	-0.94	-0.52	0.00	0.00
		average	-4.45	-5.47	-11.11	-6.68	-11.79	-5.47	-3.00	-6.70
	100	best	-6.87	-11.93	-11.92	-11.92	-11.92	-11.92	-11.92	-11.92
		worst	0.00	-0.64	-2.48	-0.64	-0.66	-0.65	0.00	-2.48
		average	-3.17	-11.47	-10.99	-11.4	-11.14	-11.47	-3.74	-11.34

Table 3: Results for test functions F4 and F8, 100 population size and various values of p_c

Func.	Crossover	Pop.size	Result	$p_c = 0.2$	$p_c = 0.5$	$p_c = 0.7$	$p_c = 0.8$	$p_c = 0.9$
F4	C3	100	best	-405.92	-405.97	-405.97	-405.97	-408.37
			worst	89.88	287.03	288.94	42.14	286.3
			average	-377.24	-381.53	-395.7	-386.66	-377.61
F8	C2	100	best	-7.0	-11.92	-7.01	-11.92	-11.81
			worst	-4.63	-0.67	0.00	-2.48	-0.72
			average	-6.93	-10.98	-6.74	-10.99	-11.22

4 Conclusion

Every good (as broad as that notion can be) defined crossover operator should achieve satisfying results for some class of the problems. Experiments conducted for the purpose of this article showed that all crossover operators tested here achieve good results. It is much more difficult to declare the winners in this experiments than the losers. The losers are definitely genetic algorithms that use no crossover operator or single-point crossover operator. The winners, at least in majority of the cases, are uniform and two-point crossover. All other crossover operators belongs to the "gray" area in-between. Depending of the properties of a problem one or another crossover operator will have better results.

Every crossover operator has its advantages and downfalls, so choosing one ultimately represents the question of someone's requirements and experiments undergone.

Further studies should include experiments with more advanced crossover methods, like those based on statistical methods, implementing more difficult test problems, and conducting experiments with larger set of crossover rates and mutation coefficients.

References:

[1] T. Baeck, D. B. Fogel and Z. Michalewicz: Handbook of Evolutionary Computation. New York, USA: Taylor & Francis Group, 1997

[2] R. A. Caruana, L.J. Eshelman and J.D. Schaffer: Representation and Hidden Bias II : Eliminating Defining Length Bias in Genetic Search via Shuffle Crossover, Proceedings of the 11th international joint conference on Artificial intelligence - Volume 1, Detroit, Michigan, USA, 1989, pp. 750–755

[3] D. Dumitrescu, B. Lazzerini, L. C. Jain and A. Dumitrescu: Evolutionary Computation. Florida, USA: CRC Press, 2000

[4] A. Ferrolho, M. Crisostomo: Genetic Algorithms: concepts, techniques and applications, WSEAS Transactions on Advances in Engineering Education, Vol. 2, 2005, pp. 12–19

[5] M. Golub: Genetski algoritam: Prvi dio. University of Zagreb, Croatia: Faculty of Electrical Engineering and Computing, 2004

[6] R. L. Haupt and S. E. Haupt: Practical genetic algorithms, second edition. New Jersey, USA: Wiley-Interscience, A John Wiley & Sons, 2004

[7] J. H. Holland: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Cambridge, USA: The MIT Press, 1992

[8] Z. Michalewitz: Genetic Algorithms + Data Structures = Evolution Programs, third edition. Berlin Heidelberg New York, USA: Springer-Verlag, 1996

[9] M. Mitchell: An Introduction to Genetic Algorithms. Cambridge, USA: The MIT Press, 1999

[10] S. Picek, M. Golub: The New Negative Slope Coefficient Measure, Proceedings of the 10th WSEAS International Conference on Evolutionary Computing, EC'09, 2009, Prag, Czech Republic, pp. 96–101

[11] S. Picek, M. Golub: Dealings with Problem Hardness in Genetic Algorithms, WSEAS Transactions on Computers, Issue 5, Volume 8, pp. 747–756

[12] S. Picek, M. Golub: On the Efficiency of Crossover Operators in Genetic Algorithms with Binary Representation, Proceedings of the 11th WSEAS International Conference on Neural Networks (NN '10), the 11th WSEAS International Conference on Evolutionary Computing (EC '10) and the 11th WSEAS International Conference on Fuzzy Systems (FS '10), Iasi, Romania, 2010, pp. 167–172

[13] S. Rana: The Distributional Biases of Crossover Operators, Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, 1999, pp. 549–556

[14] L. D. Whitley: An Executable Model of a Simple Genetic Algorithm. Foundations of Genetic Algorithms 2, 1992

[15] L. Zhonggang, Z. Liang: A Quantum-Inspired Hybrid Evolutionary Method, Proceedings of the 8th WSEAS International Conference on Applied Computer and Applied Computational Science, Hangzhou, China, 2009, pp.1422–425

[16] Genetic and Evolutionary Algorithm Toolbox for use with MATLAB Documentation, `http://www.geatbx.com/docu/fcnindex-01.html$#$P86_3059`

[17] CIXL2: A Crossover Operator for Evolutionary Algorithms Based on Population Features, `http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume24/ortizboyer05a-html/node6.html`

[18] Crossover (genetic algorithm), `http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm)`