On the Recombination Operator in the Real-Coded Genetic Algorithms

Stjepan Picek
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
Email: stjepan@computer.org

Domagoj Jakobovic
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
Email: domagoj.jakobovic@fer.hr

Marin Golub
Faculty of Electrical Engineering and Computing
Zagreb, Croatia
Email: marin.golub@fer.hr

***Abstract*—Crossover is the most important operator in real-coded genetic algorithms. However, the choice of the best operator for a specific problem can be a difficult task. In this paper we compare 16 crossover operators on a set of 24 benchmark functions. A detailed statistical analysis is performed in an effort to find the best performing operators. The results show that there are significant differences in efficiency of different crossover operators, and that the efficiency may also depend on the distinctive properties of the fitness function. Additionally, the results point out that the combination of crossover operators yields the best results.**

I. Introduction

Genetic algorithms (GAs) represent a class of the evolutionary algorithms family that has been successfully applied to various optimization processes. To enable a successful application, a perfect blend of genetic operators is often needed. Three most important operators are the selection, crossover and mutation. Here, the crossover plays an important role as an exploitation force (in fact, in Holland's version of GA the crossover had major role and mutation was generally treated as a subordinate to crossover) [1].

From the first work of Holland in 1970s genetic algorithms evolved and got more complex and naturally, the operators evolved likewise. Today, the operators are mainly divided as for binary-coded genetic algorithms, real-coded genetic algorithms and permutation based genetic algorithms. For each of these variations there are a plethora of possibilities when choosing the operators (for example, in [2] 66 binary-coded and 89 real-coded crossover operators are listed). Among so many operators it is often difficult to choose the best ones for the problem at the hand. From one perspective, the lack of publicly available research that is trying to systematically evaluate different operators is somewhat surprising. It seems that when developing a new operator it is common to compare it with just one or two well known operators on a set of just a few benchmark functions.

In this paper, we restrict our effort to the crossover operator in the real-coded genetic algorithm. Further, we compare different crossover operators on a set of benchmark functions commonly used for real-coded GAs.

When discussing a crossover operator, the first question is how important crossover operator really is. There are genetic algorithms that use extremely complex crossover schemes, and yet, there are other variations than completely exclude crossover. However, many researchers today regard the crossover operator as the most innovative and relevant operator in real-coded genetic algorithms [3] and we will adhere to that point of view. Since we determined that crossover operator is important, the next question is how to compare different crossover operators. Many of the operators are created with a special purpose at mind, so it can be regarded somewhat unfair to compare them against each other. Furthermore, in the "No Free Lunch" theorem [4] it is demonstrated that when averaged over all problems, all search algorithms perform equally. However, the "No Free Lunch" theorem operates under assumption there is no background knowledge about the problems. Since researchers rarely work on a problem that they do not know anything about, it is often possible to choose a more suitable algorithm.

We conduct our experiments on a set of well known problems [5] for which the optimum values are known, as well as a suitable representation and some of the properties of those problems. From that perspective, we feel it is possible to give a comparison of crossover operators and possibly some guidelines for their use. Since all experiments are conducted on the same set of problems, this comparison is as honest as it can be done. Naturally, for some specific problems, the crossover can behave in a completely different manner. Our experiments belong to the Multiple Algorithms - Multiple problems comparison scenario, so it is important to choose the adequate statistical procedures when evaluating the results, which is addressed in detail. The goal of our paper is to give statistically sound analysis of a set of crossover operators. As far as we know, in our experiments we compared the largest set of crossover operators for real-coded GAs up to now. Additionally, this is the first time to use Combination crossover (as defined in our paper) or to compare it against other crossover operators where those operators are building blocks of Combined crossover.

In Section 2 we present the relevant theory, Section 3 defines the experimental environment and presents the results, Section 4 gives a discussion about the results, and finally, Section 5 draws a conclusion.

II. Preliminary

A. Crossover Operator

A crossover is a process where new individuals are created from the information contained within the parents. In this paper the crossover refers to a two parent case where two individuals are selected as the parents to produce one offspring. Crossover operators can be divided into two versions for floating point strings. The first one is analogous with the crossover operator for binary-coded GAs. Only here, instead of bit values or binary coded values, alleles are floating point numbers. That kind of recombination has a disadvantage that only mutation can insert new values into the population, since recombination only gives new combinations of existing values [6]. The second one creates new allele values in the offspring that lie between the parents for each gene position [6]. Crossover operator are presented here only briefly, for further informations references are given. We believe the most important part was to write formulas for operators here, since some variations can be found in the literature describing implementation details of specific crossover operators.

If n denotes the dimensionality of the fitness function, for all crossover operators presented here holds that parents $\beta^{f_1} = \beta_1^{f_1}, \beta_2^{f_1}, ..., \beta_n^{f_1}$ and $\beta^{f_2} = \beta_1^{f_2}, \beta_2^{f_2}, ..., \beta_n^{f_2}$ form a child $\beta^s = \beta_1^s, \beta_2^s, ..., \beta_n^s$, where f_1 and f_2 are the parents' fitness values. In some operators, a parameter α is used whose value is chosen at random over $[0, 1]$ if not specified otherwise. Here, α parameter is a weighting factor.

1) Discrete Crossover: When creating an offspring β^s from parents β^{f_1} and β^{f_2}, the allele value for each gene i is given by $\beta_i^s = \beta_i^{f_1}$ or $\beta_i^s = \beta_i^{f_2}$ with equal likelihood [6].

2) Simple Arithmetic Crossover: First, a recombination point k is chosen. Then the first k float values of a random chosen parent are taken and copied into the child. The rest is the arithmetic average of parents 1 and 2 [6]. The arithmetic average is obtained via:

$$\beta^{s_1} = \alpha \cdot \beta^{f_2} + (1 - \alpha) \cdot \beta^{f_1} \tag{1}$$

3) Single Arithmetic Crossover: First, a random allele k is chosen. At that position take the arithmetic average of two parents. Other points are copied from the parents. The expression for arithmetic average is the same as for the Simple Arithmetic Crossover [6].

4) Whole Arithmetic Crossover: This is probably the most commonly used operator, which works by taking the weighted sum with the same α of two parental alleles for each gene [6]:

$$\beta^{s_1} = \alpha \cdot \beta^{f_1} + (1 - \alpha) \cdot \beta^{f_2} \tag{2}$$

5) Local Crossover: Local crossover is the same as whole arithmetic crossover, except that the value α is randomly selected for each gene location [7].

$$\beta_i^s = \alpha \cdot \beta_i^{f_1} + (1 - \alpha) \cdot \beta_i^{f_2} \tag{3}$$

6) SBX Crossover: Simulated Binary Crossover (SBX) is devised to simulate the effect of one-point binary crossover [3] [8]. Two parents β^{f_1} and β^{f_2} generate offspring β^{s_1} and β^{s_2} in the following way:

$$\beta_i^{s_1} = \frac{1}{2}\left[(1 + B_k)\,\beta_i^{f_1} + (1 - B_k)\,\beta_i^{f_2}\right] \tag{4}$$

and

$$\beta_i^{s_2} = \frac{1}{2}\left[(1 - B_k)\,\beta_i^{f_1} + (1 + B_k)\,\beta_i^{f_2}\right] \tag{5}$$

where $B_k \geq 0$ obtained from the uniform random number source $u(0, 1)$ in the following way

$$B(u) = \begin{cases} (2 \cdot u)^{\frac{1}{\eta+1}} & \text{if } u \leq \frac{1}{2} \\ \left(\frac{1}{2\cdot(1-u)}\right)^{\frac{1}{\eta+1}} & \text{if } u > \frac{1}{2} \end{cases} \tag{6}$$

7) BLX-alpha Crossover: Blend alpha crossover combines two parents β^{f_1} and β^{f_2} to generate offspring β^s by sampling a new value in the range $[min_i - I \cdot a, max_i + I \cdot a]$ at each position i. Here, min_i and max_i are smaller and larger parent values at location i. I equals $max_i - min_i$ [9]. For the α value we use 0.5. The child is generated in the following way:

$$\beta_i^s = (min_i - I \cdot a) + \alpha \cdot |(max_i + I \cdot a) - (min_i - I \cdot a)| \tag{7}$$

8) BLX-alpha-beta Crossover: Blend alpha beta crossover operator creates a new offspring by selecting a random value from the interval between the two alleles of the parent solutions β^{f_1} and β^{f_2}. The interval is increased in direction of the solution with better fitness by the factor α, and into the direction of the solution with worse fitness by the factor β [10]. The value for α is 0.75, and for β 0.25. The formula for the generation of the offspring β^s is the same as in BLX-alpha Crossover.

9) Flat Crossover: Flat crossover generates descendant β^s whose genes are randomly generated in the interval $\left[\beta^{f_1}, \beta^{f_2}\right]$ [3] [11]. This crossover is the same as BLX-alpha crossover when α = 0.

10) BGA Crossover: Two parents β^{f_1} and β^{f_2} generate offspring β^s. Let β^{f_1} be the parent with better fitness. In that case, the offspring has genes calculated in the following way:

$$\beta_i^s = \beta_i^{f_1} \pm rang_i \cdot \gamma \cdot \lambda \tag{8}$$

where λ equals

$$\lambda = \frac{\beta_i^{f_2} - \beta_i^{f_1}}{\|\beta^{f_1} - \beta^{f_2}\|} \tag{9}$$

The sign "-" is selected with 0.9 probability and $rang_i = 0.5 \cdot (b_i - a_i)$.

γ equals

$$\gamma = \sum_{k=0}^{15} \alpha_k \cdot 2^{-k} \tag{10}$$

where $\alpha_i \in 0, 1$ is randomly generated with $p(\alpha_i = 1) = \frac{1}{16}$ [3] [12].

11) Heuristic Crossover: First, take two parents β^{f_1} and β^{f_2} and assume that the first parent (β^{f_1}) has smaller value on each allele [13]. Then the offspring β^s is created as

$$\beta_i^s = \alpha \cdot (\beta_i^{f_1} - \beta_i^{f_2}) + \beta_i^{f_1} \tag{11}$$

12)Average Crossover: Two parents β^{f_1} and β^{f_2} generate offspring β^s in the following way [14]

$$\beta_i^s = (\beta_i^{f_1} + \beta_i^{f_2})/2 \tag{12}$$

13) Onepoint Crossover: In onepoint crossover the descendant is created in the same way as in the onepoint crossover for a binary-coded GA. First, choose a random point k. Two parents β^{f_1} and β^{f_2} generate offspring β^{s_1} and β^{s_2}. The first offspring copies allele values from the first parent up to the point k, and from the second parent from that point. For the second offspring the procedure is analogous [15].

14) Random Crossover: Random crossover is devised as a benchmark for testing the effectiveness of other crossover operators, so this is not a crossover operator in true sense [16]. Here, child is created by combining one individual from the parent pool and one individual with the random values:

$$\beta_i^s = (\beta_i^f + \beta_i^{random}) \tag{13}$$

B. Combination Crossover

This crossover is actually a combination of all the previous individual crossovers, where in each reproduction phase a single crossover operator is chosen uniformly at random from the pool of all available crossover operators. The reasoning behind this approach is that the evolution is able to use as many possible transitions in the search space as possible, although it may impair convergence in some cases.

Additionally, we include the case when there is no crossover operator present for the completeness of the analysis.

C. Benchmark Functions

For test functions we use functions that are available in the COCO platform (COmparing Continuous Optimisers) [5]. From that platform, 24 noise-free real-parameter single-objective benchmark functions are chosen. The naming convention for those functions adopted here is from f_1 to f_{24}. The arrangement of the functions is the same as in COCO (for instance, the function f_1 is a sphere function, and f_{24} is Lunacek bi-Rastrigin function). All functions are implemented with an arbitrary number of dimensions, which is set to 30 in this paper.

As stated in [5], the first 5 functions are separable. Functions 6-9 are with low or moderate conditioning, and from 10 to 14 are functions with high conditioning. The first 14 functions are all unimodal. The functions from 15 to 19 are multi-modal with adequate global structure and from 20 to 24 are multi-modal with weak global structure. For further informations on functions or their properties refer to [5]. We point here that for bounds we used are $[-50, 50]$, which is much larger that traditionally used in COCO ($[-5, 5]$), since we are not interested in obtaining the best solution but in finding the differences between the algorithms.

III. Environmental Settings and Results

In all the experiments, a real-coded genetic algorithm with k-tournament steady-state selection is used. In this algorithm, k individuals are randomly selected for a tournament, and the worst of those individuals is eliminated. The eliminated individual is immediately replaced with a new one by crossing the two surviving parents from the tournament and applying the mutation operator with a given probability. For the parameter k we chose $k = 3$ since that setting provides the best results from our previous experience. We did not conduct experiments with other selection procedures since the selection mechanism (or the mutation operator) were not of interest in this paper. Of course, there is a possibility that some crossover operator works better with some other selection mechanism. Since this line of research would yield exponential number of experiments (every crossover operator for every selection mechanism for every mutation operator), we chose a single mutation and a selection operator for all the experiments.

TABLE I: Parameters Used in Experiments

Parameter	Size
Tournament size k	3
Mutation probability p_m	0.3 per individual
Population size N	100
Number of runs	40
Dimensionality of the problems	30
Bounds of the problems	$[-50, 50]$
Number of fitness evaluations	1E6

As a genetic algorithm test suite the Evolutionary Computation Framework (ECF) was used [17]. ECF is a C++ framework intended for the application of any type of the evolutionary computation, developed at the University of Zagreb. The framework includes all crossover operators used in this paper.

Parameters of the genetic algorithm that are common for every round of the experiments are the following: the mutation probability is set to 0.3 per individual, population size N of 100, number of independent runs for each experiment is 30, dimensionality D of all the test problems is set to 30, and the number of fitness evaluations is set to 1E6. Bounds for all experiments are set to [-50, 50]. For all the test functions, finding the global minimum is the objective. All parameters are additionally displayed in Table I for clarity.

A. The Experiments

There are three rounds of experiments, where every round has its distinctive goal. The goal for the first round of experiments is to find the best crossover operator for all the problems, in the second round the goal is to find the best operator for unimodal problems only, and in the third round the best operator for multi-modal problems. This schedule could sound somewhat opposite from the one that should be used (i.e. first find the best operator for unimodal and multi-modal problems, and then, at the end, for all the problems) but we chose this approach in an effort to boost the power of statistical analysis. Further justification for this approach will be given in the following sections.

As a performance measure we use the average error obtained for every operator (all the test functions are implemented so the global minimum has the value of zero). For each operator and each test function, a set of 30 best solutions from 30 runs are collected and their average value is taken as the error rate. This structure of input data is in accordance with previous analysis performed over multiple algorithms and test problems [18] [19].

When conducting a statistical analysis, it is necessary to decide whether to use parametric on nonparametric statistical tests. To be able to use the parametric tests, it is necessary to check if the data satisfies the independency, normality, and heteroscedasticity conditions [18]. Parametric tests have the advantage that posses better resolution when comparing data, however, if all the necessary condition are not satisfied then using parametric tests can lead to erroneous results.

The independence of the events is obvious since there are independent runs of the algorithms. To check the normality we use the Shapiro-Wilk (S-W) test. The results obtained for the S-W test are not presented here but they show that the values do not have normal distributions. For the heteroscedasticity property we use the Levene test, which shows that heteroscedasticity is also not satisfied. Since the tests showed that the solutions are not normally distributed and the variances of the distributions of the different algorithms are not homogeneities, we rely on nonparametric statistical tests. The tests were conducted for a level of significance α of 0.05.

References regarding the statistical methods used can be found in [20] and [21]. For the statistical analysis we use R programming environment [22] and Keel [23].

B. Best Overall Performance

The goal for this round of experiments is to try to find the best crossover operator for all the problems. In this setting we have 24 problems and 16 crossover operator variations (although, two of those variations are GA without the crossover and GA with random crossover which can not really compare with real crossover operators).

Since nonparametric statistical tests have lower resolution than the parametric tests, it is important to try to follow the guidelines on the ratio of number of algorithms to the number of problems [18]. In our setting the number of algorithms (operators) is relatively large as compared to the number of test functions; still, this difference will not influence significantly the first test we conduct. The first test is the Friedman two-way analysis of variances by ranks, a well known procedure for testing the differences between more than two related samples. The goal of the Friedman test application is to try to show whether there are differences between the crossover operators. Additionally, we employ the Iman-Davenport test which is a derivation of Friedman test that produces better statistics [18]. The results for the average rankings are displayed in Table II. For each test problem the operators are given a rank from 1 to 16, based on the average error they produce on a given problem (the best operator receives the rank of 1). The average rank of each operator is calculated as the mean of all the ranks of an operator over all the test problems.

With the level of significance α of 0.05, the Friedman test reveals significant differences in crossover operators with test values of 173.59 and $p < 0.001$. The corresponding Iman-Davenport value is 21.42 and $p < 0.001$. Since there are significant differences between operators, we can use post-hoc statistical analysis that will show us where those differences are. In post-hoc analysis a comparison is made between the control crossover operator (the operator with the lowest value from Friedman test) and all the other operators. As displayed in Table II, the best operator is the Combination crossover. The first post-hoc test is the Bonferroni-Dunn test [18] [20]. This test does not have the strength in differentiating operators as the other tests used, but it has the advantage that it can be easily displayed in a graph. The interpretation of the Bonferroni-Dunn test is that the performance of two crossover operators is significantly different only if the corresponding average ranks differ by at least a critical difference [18]. The critical difference depends on the number of crossover operators and test problems - the more there are crossover operators, the critical differences will be bigger and the resolution of the Bonferroni-Dunn test lower. Because of that, in this test we used only 10 best operators based on the Friedman test. The critical difference equals 2.4236, and the complete results are displayed in Fig 1.

TABLE II: Average Rankings of the Crossover Operators

Algorithm	Ranking
Arithmetic Whole	6.791
Arithmetic Simple	8.229
Arithmetic Single	9.145
Average	8.104
BGA	6.562
Discrete	7.895
Flat	9.229
Heuristic	9.062
Onepoint	8.354
Random	14.875
SBX	6.937
No crossover	16
Local	5.791
BLX-alpha	9.229
BLX-alpha-beta	8.229
Combination crossover	1.562

Next, we applied the Bonferroni-Dunn, Hochberg, Finner and Li procedures [18] over the results of Friedman procedure. In these tests, a comparison is made between the control operator and the rest of the operators. Adjusted p-values are shown in Table III. These results indicate whether the control operator (here, Combination crossover) is better than each of the remaining operators considering level of significance α of 0.05. Results show that the Combination operator performs significantly better than any other operator. On the other hand, on a set of all test functions, no crossover operator (except for the Random and No crossover case) performs significantly worse than any other crossover operator.

C. Unimodal Problems

In this round of experiments the goal is to find the best crossover operator for test problems that are unimodal (14 in total). Since there are too many crossover operators to effectively follow the guidelines from [18] as compared to the number of available test problems, we selected the best eight (half of total number of operators) performing operators for comparison. Table IV gives the results of the average rankings of the operators.

With the level of significance of 0.05, the Friedman test shows significant differences in crossover operators with test values of 59.69 and $p < 0.001$. The Iman-Davenport value is 20.25 with a value of $p < 0.001$. Since there are significant differences, we can again employ post-hoc statistical analysis. Here, the best crossover is the Combination crossover so we use it as the control operator. In the Bonferroni-Dunn test the

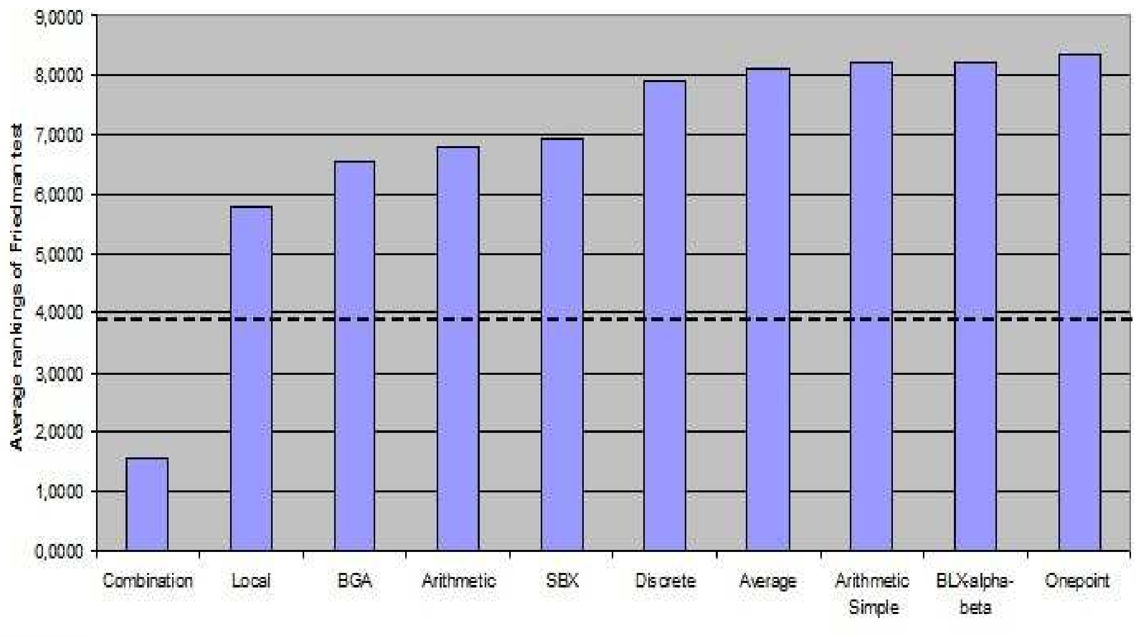

Fig. 1: Bonferroni-Dunn's test (CD = 2.42, control operator: Combination crossover)

TABLE III: Post-hoc comparison (control operator: Combination crossover)

Algorithm	unadjusted p	p_{Bonf}	$p_{Hochberg}$	p_{Finner}	p_{Li}
No crossover	0	0	0	0	0
Random	0	0	0	0	0
BLX-alpha	0	0.000001	0	0	0
Flat	0	0	0	0	0
Arithmetic Single	0	0.000001	0	0	0
Heuristic	0	0.000001	0	0	0
Onepoint	0.000001	0.000012	0.000007	0.000002	0.000001
Arithmetic Simple	0.000001	0.000018	0.00009	0.000002	0.000001
BLX-alpha-beta	0.000001	0.000018	0.00009	0.000002	0.000001
Average	0.000002	0.000029	0.000012	0.000003	0.000002
Discrete	0.00004	0.000061	0.00002	0.000006	0.00004
SBX	0.000092	0.001379	0.000368	0.000115	0.000092
Arithmetic Whole	0.000142	0.002129	0.000426	0.000164	0.000142
BGA	0.000275	0.004121	0.000549	0.000294	0.000275
Local	0.00209	0.031346	0.00209	0.00209	0.00209

TABLE IV: Average Rankings of the crossover operators/unimodal problems

Algorithm	Ranking
Arithmetic Whole	3.6071
Arithmetic Simple	5.9643
BGA	3.6786
SBX	4.5357
Local	3.3214
Arithmetic Single	7.75
Average	5.6071
Combination crossover	1.5357

critical difference is 3.95 and the results are displayed in Fig. 2.

From Figure 2 we can see that it cannot be said whether Combination crossover is significantly better than Local, Whole Arithmetic or BGA crossover. The results from the post-hoc statistical analysis are presented in Table V. The table shows that the Combination operator performs better than any other operator for unimodal problems, except for the Local crossover operator.

D. Multi-modal Problems

Here the goal is to find the best crossover for multi-modal problems (10 problems). Again we chose the best eight crossover operators to conduct the analysis, for which the average rankings are displayed in Table VI. Here Friedman test shows significant differences with value 30.96 and $p < 0.001$, and Iman-Davenport with 7.1 and $p < 0.001$. The control algorithm is the Combination crossover.

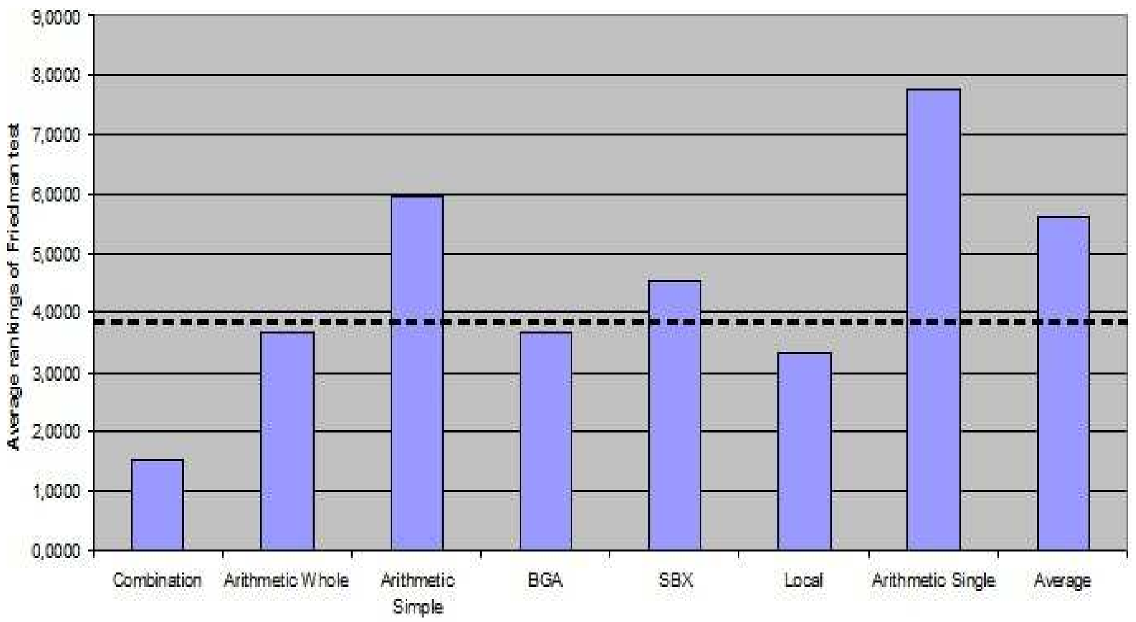

Fig. 2: Bonferroni-Dunn's test (CD = 2.42, control operator: Combination crossover, unimodal problems)

TABLE V: Post-hoc comparison (control operator: Combination crossover)

Algorithm	unadjusted p	p_{Bonf}	$p_{Hochberg}$	p_{Finner}	p_{Li}
Arithmetic Single	0	0	0	0	0
Arithmetic Simple	0.000002	0.000012	0.00001	0.000006	0.000002
Average	0.000011	0.000077	0.000055	0.000026	0.000012
SBX	0.001194	0.008356	0.004775	0.002088	0.00126
BGA	0.020638	0.144463	0.050521	0.028773	0.021344
Arithmetic Whole	0.02526	0.176822	0.050521	0.029408	0.026001
Local	0.053757	0.376297	0.053757	0.053757	0.053757

TABLE VI: Average Rankings of the crossover operators/multi-modal problems

Algorithm	Ranking
BLX-alpha	4.9
Flat	5.1
BLX-alpha-beta	3.5
Heuristic	4.5
SBX	6.9
Onepoint	5
Discrete	4.9
Combination crossover	1.2

Bonferroni-Dunn analysis is displayed in Fig. 3 with a critical difference of 3.62. We can see that the Combination crossover operator performs better than any of the algorithms except for the BLX-alpha-beta crossover. Post-hoc statistics is displayed in Table VII. After post-hoc analysis, we can see that the results obtained from the Bonferroni-Dunn test are confirmed, i.e. the Combination crossover performs better than any of the other operators for multi-modal functions, except for the BLX-alpha-beta crossover.

IV. Discussion

There are too many possible combinations to try if one would want to test every combination of crossover, selection and mutation operators. Because of that, we test the performance of only one operator - the crossover. We performed three rounds of experiments in an effort to conclude which crossover operators perform better. In the first round when the goal was to find the best crossover operator for all the test functions, the results show that the Combination crossover performed the best. We are not aware that a comparison of this kind of combined crossover with separate operators has been presented before. Results that are obtained are expected from one perspective, since in the Combination operator we have all the other crossover operators, so definitely there are schemes that can lead to good results. From the other perspective, since there is no weighting factor when choosing a crossover operator from the pool of operators, it could be possible that some operators actually lead the search away from optimal values. This can be especially so in case of Random crossover (of course, Random crossover can be regarded as some instantiation of ultimate mutation operator). It is hard to choose from the other good operators (ten best ones) which one would be

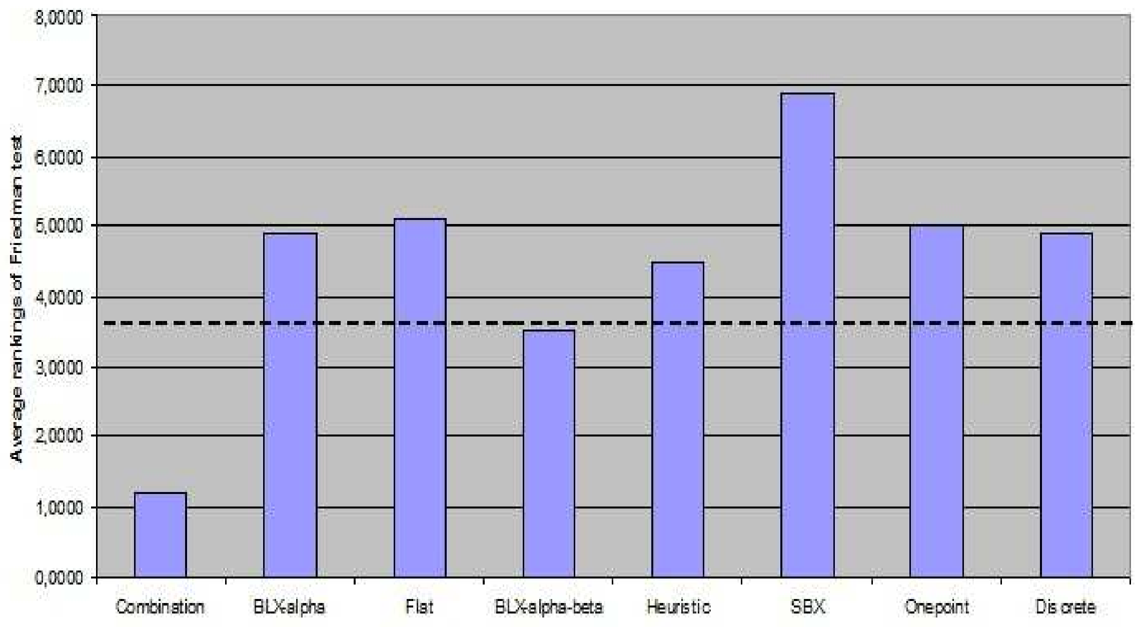

Fig. 3: Bonferroni-Dunn's test (CD = 2.42, control operator: Combination crossover, multi-modal problems)

TABLE VII: Post-hoc comparison (control operator: Combination crossover)

Algorithm	unadjusted p	p_{Bonf}	$p_{Hochberg}$	p_{Finner}	p_{Li}
SBX	0	0.000001	0.000001	0.000001	0
Flat	0.000371	0.002594	0.002193	0.001296	0.000542
Onepoint	0.000523	0.003658	0.002193	0.001296	0.000542
BLX-alpha	0.000731	0.005118	0.002193	0.001296	0.000758
Discrete	0.000731	0.005118	0.002193	0.001296	0.000758
Heuristic	0.002591	0.018139	0.005183	0.003023	0.00268
BLX-alpha-beta	0.035764	0.250346	0.035764	0.035764	0.035764

the second best, but the Local crossover and BGA are quite close.

In the second round of the experiments we tried to find the best algorithm for unimodal problems. Again, the Combination crossover exhibited the fastest convergence. After this operator, the best ones were the Whole arithmetic and the Local crossover. It is worth to be reminded here that those two operators are very similar (the only difference is in the weighting factor). Successfulness of the Whole arithmetic crossover is of no surprise, since it is the most commonly used crossover operator in real-coded GAs and that shows it endured many comparisons and constantly performed good. The success of the Local crossover is somewhat more surprising since it is not so common in use. Still, since they are quite similar it is easy to change from one version to another. That similarity can also be regarded as a sanity check: if one operator performs good, then the other should also perform similarly.

The third round of experiments concentrates on multi-modal problems. The same as in round one and two, the Combination crossover is again the one with the best performance. The second best is BLX-alpha-beta; this can be expected since the nature of that operator is quite adapt for multi-modal cases. It is quite important to observe that no variation of arithmetic crossovers (including Local) reached the top eight operators. This should strongly indicate that those kind of operators are not suitable for multi-modal problems.

It should be noted that the obtained results are also dependent on the chosen maximum number of evaluations, which was 1E6 in the experiments. In a different setting, e.g. with different number of dimensions and where constraints may dictate a different termination criteria, the outcomes may vary.

A. *Further Research*

There are several lines of further research we are currently investigating. First, we are interested in the performance of crossover operators when coupled with various selection and mutation operators. Second, we plan to combine some of the crossover operators with Taguchi [24] method which can select better genes that undergo crossover process and by it, enhance the genetic algorithm. We are currently working on a procedure that can help us differentiate various test problems based on their fitness landscapes [25]. Our goal is to check whether there is a connection between a fitness landscape type and the performance of a crossover operator.

Since the Combination crossover operator performed best in all the experiments, it would be interesting to investigate the influence of each crossover operator in this combined fashion. Our intention is to repeat all the experiments but with a reduced number of best crossover operators (from each category). After that, we will add operators one at a time to check what is the optimal combination.

V. Conclusion

In this work an exhaustive search was performed to find the best crossover operator on a set of well-known optimization problems. The results show that there are significant differences between crossover operators. However, a combination of individual crossover operators, where an operator is chosen at random each time the crossover is needed, performed better than any other single crossover operator in most of the experiments. If we neglect this Combination crossover from the analysis, we can see that the differences between operators are much smaller and in some cases there is no statistically significant difference. From the obtained results we can form a recommendation to use a combination of crossover operators instead of a single operator. Naturally, there remains the open question of what is the smallest subset of crossover operators that yields significantly better results. In any case, since many frameworks have more than one crossover operator implemented, there should be no problems in using such a combination.

References

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

[2] T. D. Gwiazda, *Genetic Algorithms Reference*. Tomasz Gwiazda, 2006.

[3] D. Ortiz-Boyer, C. Hervas-Martinez, and N. Garcia-Pedrajas, "Improving crossover operators for real-coded genetic algorithms using virtual parents," *Journal of Heuristics*, no. 13, pp. 265–314, 2007.

[4] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, April 1997.

[5] N. Hansen, S. Finck, R. Ros, and A. Auger, "Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions," Tech. Rep. RR-6829, 2009.

[6] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg New York, USA, 2003.

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

[8] K. D. R. B. Agrawal, "Simulated binary crossover for continuous search space," Tech. Rep., 1994.

[9] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *FOGA*, L. D. Whitley, Ed. Morgan Kaufmann, 1992, pp. 187–202.

[10] M. Takahashi and H. Kita, "A crossover operator using independent component analysis for real-coded genetic algorithms," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, 2001, pp. 643–649.

[11] N. J. Radcliffe, "Equivalence class analysis of genetic algorithms," pp. 183–205, 1991.

[12] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm i. continuous parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 25–49, Mar. 1993.

[13] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 205–218.

[14] T. Nomura, "An analysis on crossovers for real number chromosomes in an infinite population size," in *Proceedings of the Fifteenth international joint conference on Artifical intelligence - Volume 2*, ser. IJCAI'97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 936–941.

[15] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (3rd ed.)*. London, UK, UK: Springer-Verlag, 1996.

[16] T. Jones, "Crossover, macromutation, and population-based search," in *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995, pp. 73–80.

[17] D. Jakobovic and et al., "Evolutionary computation framework," Jan. 2013. [Online]. Available: http://gp.zemris.fer.hr/ecf/

[18] S. Garcia, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms? behaviour: a case study on the CEC 2005 special session on real parameter optimization," *Journal of Heuristics*, no. 15, pp. 617–644, 2009.

[19] S. Picek, M. Golub, and D. Jakobovic, "Evaluation of crossover operator performance in genetic algorithms with binary representation," in *ICIC (3)*, 2011, pp. 223–230.

[20] D. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 4th ed. Chapman and Hall/CRC, 2007.

[21] T. Bartz-Beielstein, *Experimental Research in Evolutionary Computation: The New Experimentalism*. Springer, New York, USA, 2006.

[22] "The R project for statistical computing," 2013. [Online]. Available: http://www.r-project.org/

[23] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. Garca, L. Snchez, and F. Herrera, "Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, pp. 255–287, 2011.

[24] J.-T. Tsai, T.-K. Liu, and J.-H. Chou, "Hybrid taguchi-genetic algorithm for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 4, pp. 365–377, August 2004.

[25] E.-G. Talbi, *Metaheuristics - From Design to Implementation*. Wiley, 2009.