Does Constraining the Search Space of GA Always Help? The Case of Balanced Crossover Operators

Luca Manzoni[1], Luca Mariot[1], and Eva Tuba[2]

[1]Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy ,
{luca.manzoni luca.mariot}@unimib.it

[2]Faculty of Informatics and Computing, Singidunum University, Danijelova 32, 11000 Belgrade, Serbia ,
etuba@ieee.org

July 31, 2019

Abstract

In this paper, we undertake an investigation on the effect of balanced and unbalanced crossover operators against the problem of finding non-linear balanced Boolean functions: we consider three different balanced crossover operators and compare their performances with classic one-point crossover. The statistical comparison shows that the use of balanced crossover operators gives GA a definite advantage over one-point crossover.

Keywords– genetic algorithms, crossover operators, balanced bitstrings, Boolean functions, orthogonal arrays, bent functions

1 Introduction

Crossover operators play a crucial role in Genetic Algorithms (GA). In the case of binary strings, there exist several classes of combinatorial optimization problems whose feasible solutions must contain a specified number of ones, which are difficult to handle for classical crossover operators. A way to address this problem is to design recombination operators that *preserve* the Hamming weight of the bitstrings: the *balanced crossover operators*. In the literature, the motivation supporting the use of such operators is the reduction of the search space [5]. However, it is not clear whether balanced crossover operators actually bring any advantage to GA working with fixed Hamming weight bitstrings. Some works in the literature [8, 9] performed a comparison with non-parametric test on classic

crossover operators, but did not consider balanced operators. The aim of this paper is to begin closing this gap: we consider three balanced crossover operators in our investigation for finding non-linear balanced Boolean functions.

2 Balanced Crossover Operators

Previous work on the design of balanced crossover operators includes [7, 1, 6]. In the following, $\mathbb{F}_2$ is $\{0, 1\}$, a *bitstring* of length $n \in \mathbb{N}$ is a binary vector x of n components, and we say that x is balanced when it is composed of an equal number of zeros and ones. We now describe the crossover operators adopted in our experiments. Each of these operators is based on a different encoding for the chromosome of a balanced solution.

Counter-Based Crossover. This crossover uses the *binary vector coding*. With it, the simplest way to design a balanced operator is to randomly select bit-by-bit the allele from the first or the second parent to be copied in the offspring (as in uniform crossover), and use counters to keep track of the multiplicities of ones and zero in the child. When one of the two counters reaches the prescribed threshold, the child is filled the complementary value. Millan et al. [7] were the first who proposed this operator to evolve nonlinear balanced Boolean functions. Later works [3, 4, 5] adapted this operator to similar optimization problems.

Map of Ones Crossover. The *map of ones* of x is the vector $q = (q_1, \cdots, q_k)$ where for all $i \in \{1, \cdots, k\}$ it results that q_i has value one. One can notice that the only constraint in the map of ones is that there cannot be duplicate positions in the vector. Thus, given two bitstrings represented by their maps of ones, our crossover operator is aware of the common positions between them, in order to avoid duplications.

Zero Lengths Crossover. Given the bitstring x, the *zero lengths coding* of x is the vector $r = (r_1, \cdots, r_{n-k+1})$ which lists the *distances between consecutive ones* in x. The zero lengths vector of a bitstring of length n and Hamming weight k is valid if and only if the sum of the components in the vector equals k. Our crossover operator based on the zero lengths representation thus controls the sum of the run lengths of zeros in the offspring.

3 Experiments

Nonlinear Balanced Boolean Functions A *Boolean function* of $n \in \mathbb{N}$ variables is a map $f : \mathbb{F}_2^n \to \mathbb{F}_2$. The *truth table* is basically a binary vector of length 2^n that specifies for each input vector $x \in \mathbb{F}_2^n$ the output value of $f(x)$. A Boolean function is called *balanced* if its *truth table* is composed of an equal number of ones and zeros. The *nonlinearity* of Nl a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as the minimum Hamming distance of its truth table from the set of truth tables of all linear functions. We refer the reader to [7] about the optimization problem of finding highly nonlinear Boolean functions, and its relevance to cryptography. Given the truth table bitstring of a Boolean function

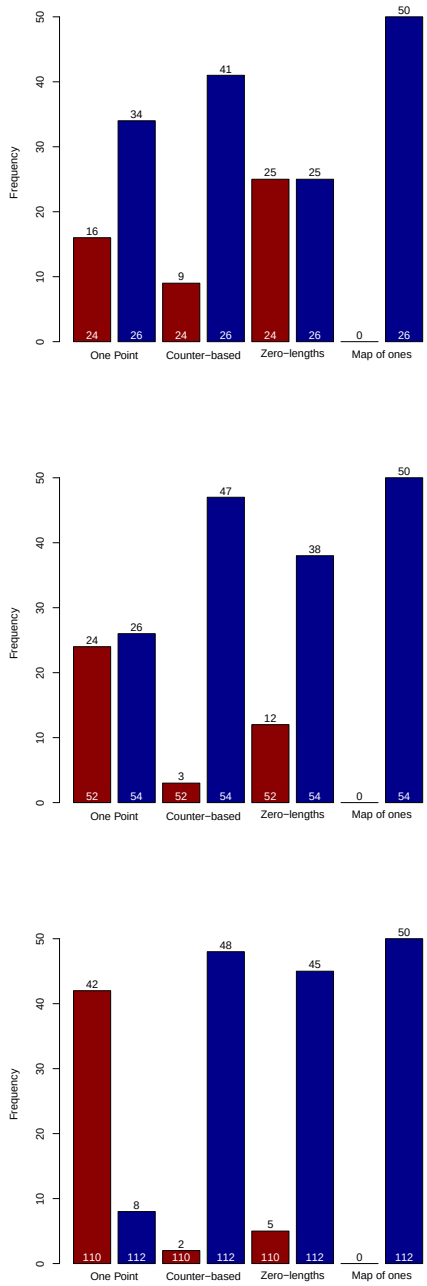

Figure 1: The results for the balanced Boolean functions problem for 6 (up), 7 (center), and 8 (bottom) variables.

$f : \mathbb{F}_2^n \to \mathbb{F}_2$ of n variables, in our experiments the fitness of f is computed with the following function: $\text{fit}_1(f) = Nl(f) - UNB(f)$ where $UNB(f)$ is the *unbalancedness penalty factor* which punishes the deviation of f from being a balanced function. The objective of our GA, in particular, is to *maximize* $\text{fit}_1(f)$.

Experimental Settings We use a *steady state* GA, where a single pair of parents is drawn from the current population at each iteration. For selection, we employed a *deterministic tournament* operator where the best two out of t randomly sampled individuals are selected for crossover. Our GA generates a single child for each selected pair of parents. The mutation operator depends on the type of crossover: when one-point crossover is used, a classic bit-flip mutation operator is applied on the generated child. With balanced crossover operators a simple *swap-based* mutation operator is used. For one-point crossover, the population is initialized at random, while for balanced crossover operators all the individuals in the initial population are balanced. The GA uses a worse-replacement elitist strategy: if the child has a better fitness value than any of its two parents, then the worse individual in the population is replaced by it. For each problem instance, we ran our GA with each of the four crossover operators for $R = 50$ experimental runs. Hence, we performed a total of $3 \cdot 4 \cdot 50 = 600$ experiments. Each of them used a population size of $P = 50$ individuals, tournament size $t = 3$ and mutation probability $p_m = 0.2$, and stopped after $fit = 500000$ fitness evaluations. To compare the results we employed the *Mann-Whitney-Wilcoxon test* [2] with significance value α set to 0.01.

Results The results of the experiments are summarized in Figure 1. For 6 variables, the map of ones crossover seems to produce the best results, with all fitness values obtained being equal to 26 (recall that this is a maximization problem, so higher values are better). This is evident also in the statistical tests, with a significant difference between the distribution of the "map of ones" results and all the other methods, with p-values of $1.4 \cdot 10^{-5}$, 0.0018, and $9.5 \cdot 10^{-9}$ when compared to the one point, counter-based, and zero-lengths crossover, respectively. Similar results hold for 7 variables, where the map of ones and the counter-based operators perform better than one-point crossover (p-values of $2.6 \cdot 10^{-6}$ and $2.3 \cdot 10^{-8}$, respectively). The map of ones crossover also performs better than the zero-lengths crossover (p-value of 0.0002), but no other comparison of the results gives a statistically significant difference. The results are different in the case of 8 variables, with one-point crossover resulting in a statistically significant difference with all other operators (in all cases the p-values are less than 10^{-12}). Therefore, it appears as if the map of ones crossover is, on this problem, the best performer, but its advantage when the problem size increases is not preserved with respect to the others balanced crossovers. When the problem size increases, the inability for one point crossover to preserve balancedness is a serious drawback, making it the worst performer for 8 variables.

References

[1] J. Chen and J. Hou. A combination genetic algorithm with applications on portfolio optimization. In *IEA/AIE*, volume 4031 of *Lecture Notes in Computer Science*, pages 197–206. Springer, 2006.

[2] S. García, 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. *J. Heuristics*, 15(6):617–644, 2009.

[3] L. Mariot and A. Leporati. A genetic algorithm for evolving plateaued cryptographic boolean functions. In *Theory and Practice of Natural Computing - Fourth International Conference, TPNC 2015, Mieres, Spain, December 15-16, 2015. Proceedings*, pages 33–45, 2015.

[4] L. Mariot, S. Picek, D. Jakobovic, and A. Leporati. Evolutionary algorithms for the design of orthogonal latin squares based on cellular automata. In *GECCO*, pages 306–313. ACM, 2017.

[5] L. Mariot, S. Picek, D. Jakobovic, and A. Leporati. Evolutionary search of binary orthogonal arrays. In *PPSN (1)*, volume 11101 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2018.

[6] T. Meinl and M. R. Berthold. Crossover operators for multiobjective k-subset selection. In *GECCO*, pages 1809–1810. ACM, 2009.

[7] W. Millan, A. J. Clark, and E. Dawson. Heuristic design of cryptographically strong balanced boolean functions. In *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 489–499. Springer, 1998.

[8] S. Picek, M. Golub, and D. Jakobovic. Evaluation of crossover operator performance in genetic algorithms with binary representation. In *ICIC (3)*, volume 6840 of *LNCS*, pages 223–230. Springer, 2011.

[9] S. Picek, D. Jakobovic, and M. Golub. On the recombination operator in the real-coded genetic algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*, pages 3103–3110, 2013.