2018 IEEE Latin American Conference on Computational Intelligence (LACCI)
Guadalajara, Jalisco, Mexico, November 7-9, 2018

A Biased Random-Key Genetic Algorithm for Regression Test Case Prioritization

Pablo Carballo, Pablo Perera, Santiago Rama and Martín Pedemonte, *Member, IEEE*
Instituto de Computación
Universidad de la República
Montevideo, Uruguay
Emails: {pablo.carballo,pablo.perera,santiago.rama,mpedemon}@fing.edu.uy

***Abstract*—The Test Case Prioritization Problem (TCPP) is a real-world problem that arises in regression testing. It lies in finding an ordering of the test cases of a test suite, such that test cases ordered first should be run first. The classical approach to solve this problem employing GAs is to use permutational encoding, but it requires specific operators to keep the feasibility of the solutions. The Biased Random-Key Genetic Algorithm (BRKGA) follows a different philosophy for dealing with permutations, using a string of real numbers and a decoder for computing the permutation. In this paper, we propose a BRKGA for solving the TCPP. The experimental evaluation on eleven instances of seven real-world programs shows that BRKGA is able to outperform two different permutational encoding based GAs (with order and cycle crossover operators), and that it has at least a similar performance than another permutational encoding based GA (with partially-mapped crossover) and a GA from a previous work specially conceived for tackling the TCPP.**

***Index Terms*—Regression testing, Search-based software engineering, Biased random-key genetic algorithm, Test case prioritization problem.**

I. Introduction

Software testing is a key activity of the software development process. The goal of software testing is evaluating the quality of a piece of software through its execution. It involves the use of test cases that are specially designed for exercising at least one feature of the piece of software under evaluation.

Regression testing is an activity conducted to ensure that changes made to a piece of software do not introduce new errors. During the evolution of a piece of software, the software usually grows in size and complexity. As a consequence, new test cases are continually added to the test suite to validate the latest modifications. For this reason, the execution of the entire test suite is often impracticable. Several different approaches have been proposed in order to reduce the effort devoted to regression testing [1].

In particular, the Test Case Prioritization Problem (TCPP) is a $\mathcal{NP}$-hard real-world software testing problem that arises in regression testing. It is based on finding an ordering of the test cases according to some specific criteria, such that test cases ordered first should be run first [1]. In this approach, all test cases can be executed but the testing process could be stopped at some arbitrary point. As a consequence, if the test process is stopped, the test cases that have already been executed are those that maximize the specific criteria used for ordering them. In order to tackle this problem properly, we make use of Genetic Algorithms (GAs) [2].

978-1-5386-4626-7/18/$31.00 © 2018 IEEE

Since real-world software programs are composed of thousands of lines of code, and consequently the test suites used involve thousands of test cases, exact algorithms are usually discarded for such problems. Metaheuristics [3] have emerged as an alternative to exact methods because they are able to provide very accurate solutions in a reasonable amount of time. The application of search-based optimization techniques, specially metaheuristics and GAs, to software engineering problems has grown substantially in the last years, which is known as search-based software engineering (SBSE) [4].

Genetic algorithms are one of the most popular types of Evolutionary Algorithm. In GAs, individuals are usually represented using binary strings. Binary encoding was originally considered as the standard representation of GA. In fact, binary strings can be used for encoding integers, reals, graphs or sets. Binary strings are also easy to handle, which greatly facilitates the design of crossover and mutation operators. However, other encodings are also possible like real encoding [5] or permutational encoding [6].

Since the goal of the TCPP is to find the best ordering of the test cases according to some specific criteria, the most natural approach to solve this problem employing GAs is to use permutational encoding. However, this approach requires the use of specific operators that are designed to keep the feasibility of the solutions because the resulting sequence cannot have repeated values.

The Biased Random-Key Genetic Algorithm (BRKGA) [7] has gained great interest in recent years. BRKGA is an extension of the idea proposed by Bean [8] for solving sequencing problems with GAs. In BRKGA, as well as in Bean's proposal, individual are represented using a string of real numbers in the [0, 1] interval. The algorithm uses a decoder that maps an individual to a solution of the optimization problem. As a consequence, the algorithm searches freely in the continuous n-dimensional unit hypercube (without any feasibility restriction), and uses the decoder to calculate the corresponding solution in the solution space. In particular, in the case of a permutation, the decoder sorts the string of real values and uses the indices of the sorted values to represent the ordering of the elements.

In this work, we study the use of a BRKGA for solving

the TCPP. One of our goals is to make a comparative analysis between BRKGA and classical permutational encoding based GAs. With this in mind, we make a comparative experimental study on the numerical performance of BRKGA and three different GAs with permutational encoding, which use different crossover operators. But since it is also our goal to analyze whether BRKGA is effective for the problem, we also consider in the experimental study a GA from the literature, which was specially designed for the problem at stake.

This article is organized as follows. Section II introduces the TCPP and discusses the related paper in the literature. Then, in Section III, classical crossover and mutation operators for the permutational encoding and BRKGA are introduced. The details of the empirical study and the analysis of the experimental evaluation results are discussed on IV. Finally, in Section V, we outline the conclusions of this work and suggests future research lines.

II. Test Case Prioritization Problem

Three different problem formulations have been mainly studied for test suite reduction in regression testing [1]. The Test Suite Minimization Problem [9]–[11] consists in reducing a large test suite by removing redundant test cases, ensuring that a set of test goals are satisfied. The goal is to find a reduced test suite that minimizes the amount of resources required for its execution and that covers a set of test goals of the piece of software that is being tested. Another approach is known as the Test Case Selection Problem, which lies in selecting a subset of the suite based on which test cases are relevant for testing the changes of the piece of software between the new version and the previous version.

In the present work, we adopt the problem formulation known as Test Case Prioritization Problem (TCPP). In this approach, the goal is to order the test cases of the suite in such a way that desirable properties, such as the number of failures detected or code coverage, are early maximized. Although it is assumed that eventually the whole test suite can be executed, in case that the testing process is stopped (for instance because the time available to run the test cases is over), the test cases that early maximize the desirable property are already executed.

This problem was formally defined by Rothermel et al. [1], [9] as follows. Let T be a test suite, PT be the set of permutations of T, and f be a function from PT to real numbers ($f : PT \rightarrow \mathbb{R}$). f is the goal function that assigns a value to any ordering of the test cases for the desirable property that has to be maximized. The TCPP consist in finding $T' \in PT$ such that $\forall T''$ with $T'' \in PT$ and $T'' \neq T'$ it holds that $f(T') \geq f(T'')$.

Several different goals have been proposed in the literature [12], such as the rate of fault detection (test cases are ordered according to the number of faults detected in previous executions of the test suite), the rate of critical fault detection (similar to the first goal but taking into account the severity of the faults), the rate of code coverage and a cost associated to the test case execution (the cost of a test case can be measured in terms of the time required to configure and execute it or the economic cost associated to the test).

In this work, the goal is to maximize the rate of code coverage following the approach proposed by Li et al. [13]. In other words, the test cases that produce a greater code coverage are executed first. The use of this goal is based on the assumption that maximizing code coverage increases the probability of maximizing fault detection. In particular, we use the Average Percentage Coverage (APC) metric for measuring the effectiveness of the prioritization of test cases. This metric measures the rate at which the test cases covers a set of test goals.

The APC metric is defined as follows. Given a test suite T that contains n test cases that cover m test goals. Let TC_i be the first test case in the prioritized test suite T' that covers test goal i. The APC for T' is calculated as in (1).

$$APC = 1 - \frac{TC_1 + TC_2 + \ldots + TC_m}{nm} + \frac{1}{2n} \tag{1}$$

The APC metric measures the weighted average of test goal coverage of a prioritized test suite. The values of APC range between 0 and 1 (or equivalently as a percentage between 0 and 100), a higher value indicates a faster (i.e., better) coverage rate.

Li et al. [13] have also evaluated a GA for the TCPP. The GA proposed by Li et al. uses permutational encoding, the OX crossover operator (explained in Section III-A), the swap mutation (explained in Section III-A) and a selection method that uses rank-based fitness assignment. The authors have also evaluated three different greedy algorithms and a local search algorithm. The experimental results show that there are no statistically significant differences between the GA proposed and the best performing greedy algorithm for the instances considered in the experiments. However, the analysis also shows that there are situations where the entire ordering has to be considered, and in these scenarios the GA is the best option due to their generality.

GAs have also been used for the TCPP using historical records from the last regression testing [14] and test-points coverage [15], and in other scenarios as the design of automated test plan in agile environments [16]. In the last years, the use of multi-objective GAs to solve the multi-objective formulation of the test case prioritization problem [17]–[19] has also gained popularity. Several conflicting objectives have been studied in the literature as minimizing the execution time, maximizing fault detection and the severity of the faults detected, maximizing code or features coverage, etc.

III. Genetic Algorithms for Permutational Optimization Problems

In this section, first we present classical crossover and mutation operators for the permutational encoding, and then we describe the BRKGA.

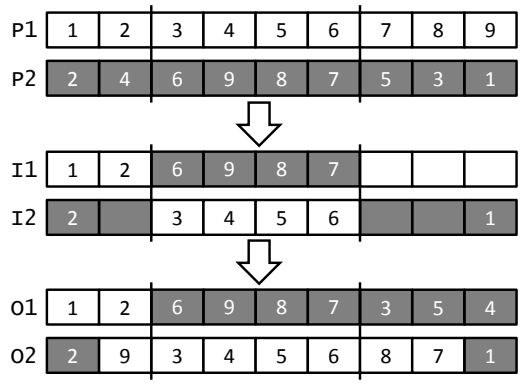

Fig. 1. Partially-Mapped Crossover Example.

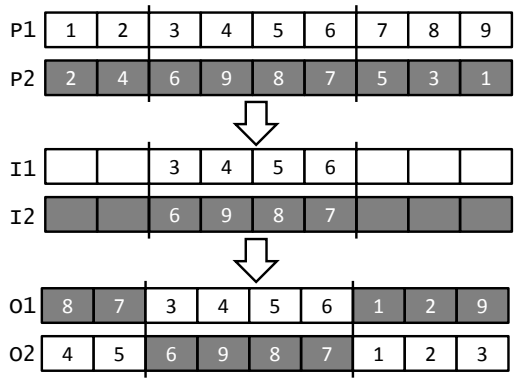

Fig. 2. Order Crossover Example.

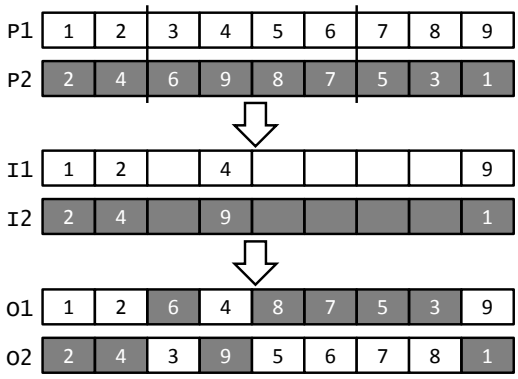

Fig. 3. Cycle Crossover Example.

A. Permutational Encoding Operators

In this subsection we introduce the partially-mapped, order, and cycle crossover operators, and the swap mutation operator for permutational encodings.

In the partially-mapped crossover (PMX) [20], two random cut points are selected on the parents, and the segments between the two cut points are exchanged. The segments exchanged are also used as a mapping between parents to replace elements that are repeated in the descendant chromosomes by those elements that were in the chromosome before the exchange.

Fig. 1 shows an example of the PMX operator. The mappings are $3 \leftrightarrow 6$, $4 \leftrightarrow 9$, $5 \leftrightarrow 8$ and $6 \leftrightarrow 7$. The mapping is used for determining the values that replace the values that are repeated, for instance, in the second offspring 9 replaces 4 and 7 replaces 3 (3 is mapped to 6 but 6 is already in the second offspring, so 6 is mapped to 7).

The order crossover (OX) [21] also selects two random cut points, but in this case the segments between cut points are copied to the offsprings. After this, the values from the one parent are copied to the other in the same order omitting repeated values, starting from the second cut point.

Fig. 2 shows an example of the OX operator. The sequence of values in the second parent from the second cut point is $1-2-9-8-7$ (after the removal of $5-3-4-6$, which are already in the first offspring). These values are placed in the first offspring beginning from the second cut point.

The cycle crossover (CX) [22] is based on the identification of a cycle shared between the two parent. The cycle is used as the genetic material that the offspring shares with one of his parents, and the rest of the positions are completed with the values that are missing from the other parent in order to generate the new permutation.

Fig. 3 shows an example of the CX operator. The cycle shared between the two parent is $1-2-4-9-1$. These values are copied from parent 1 to offspring 1 and from parent 2 to offspring 2. Then, the rest of the values of each offspring are copied from the other parent.

Finally, the swap mutation operator selects two random points on the chromosome, and the values of these points are exchanged, leaving the values of the rest of the positions unmodified.

B. Biased Random-Key Genetic Algorithm

The Biased Random-Key Genetic Algorithm (BRKGA) [7] follows a different philosophy for dealing with permutations, using a string of real numbers in the [0, 1] interval for representing the individuals. The algorithm also uses a deterministic decoder, which calculates for any individual the corresponding solution of the problem being solved for which the fitness value can be calculated. The decoder used for permutations sorts the string of real values and uses the indices of the sorted values to represent the permutation.

The algorithm of BRKGA is described next. Initially, p individuals are randomly generated, i.e., each allele is generated at random in the real interval [0, 1]. The BRKGA evolves the population over a number of generations. At each iteration, after computing the fitness using the decoder, the population is partitioned into two groups: a set of p_e elite individuals with the best fitness values, and the rest of the $p - p_e$ individuals of the population.

The new population is generated using three different mechanisms: elitism, mutants and mating. The BRKGA uses elitism, i.e., all elite individuals of a generation are copied without changes to the next generation, assuring that highly fitted genetic material is preserved in the population. BRKGA

does not use a mutation operator, but instead introduces p_m mutants into the population in order to avoid getting stuck in local optima. The mutants are new individuals that are generated in the same way that the initial population. The rest of the population ($p - p_e - p_m$) for completing the next generation of the population is generated by mating. The mating in BRKGA selects at random one parent from the set of elite individuals and the other parent from the set of non-elite individuals, and uses the Parameterized Uniform Crossover (PUX) [23]. In the PUX operator, the value of each allele of an offspring is selected with a given probability ρ_e from one parent and with $1 - \rho_e$ from the other parent. In particular in the BRKGA, ρ_e represents the probability of inheriting the allele from the elite parent and $\rho_e \geq 0.5$.

IV. Experimental Analysis

This section reports on the results of the experiments performed to evaluate five different GAs for the TCPP. First, we present the experimental setup. Then, we detail the experimental results and discussion.

A. Experimental Setup

The subject programs used in this work are real world programs that belong to the Siemens benchmark suite [24]. The Siemens suite is publicly available at the SIR website [25]. The suite includes an aircraft collision avoidance system (`tcas`), a statistic computation program (`totinfo`), two priority schedulers (`schedule` and `schedule2`), two lexical analyzers (`printtokens` and `printtokens2`) and a program that performs pattern matching and substitution (`replace`). Table I presents the Siemens programs including the number of test cases, the number of test goals and the number of source lines of code (LOCs). In this case, the test goals came from structural coverage. Test goals are already included in the source codes available in the repository.

TABLE I
Subject programs used in the evaluation.

Subject	Test Pool Size	Test Goals	LOCs
`tcas`	1608	54	162
`totinfo`	1052	117	346
`schedule`	2650	126	299
`schedule2`	2710	119	287
`printtokens`	4130	195	378
`printtokens2`	4115	192	366
`replace`	5542	208	514

In a previous work [11], 1000 test suites were randomly generated for each subject program following the methodology used by several authors [9]. Even though the suites were originally generated for the TSMP, they can also be used for the TCPP. We have selected eleven instances for each of the subject programs for the experiments. Table II presents the instances used in the experimental evaluation including the minimum and maximum number of test cases of the 1000 instances, and the number of test cases of the eleven instances selected.

TABLE II
Instances of the experimental evaluation.

Subject	Min	Max	Test cases of the instances used										
`tcas`	3	81	3	8	16	24	33	41	49	59	67	73	81
`totinfo`	4	173	4	20	37	51	72	90	105	123	142	156	173
`schedule`	3	149	3	14	32	47	63	78	92	108	123	137	149
`schedule2`	3	143	3	14	29	47	61	72	87	100	113	127	143
`printtokens`	8	189	8	23	43	61	81	96	115	131	150	168	189
`printtokens2`	7	183	7	23	38	55	74	95	112	129	148	166	183
`replace`	6	257	6	31	55	82	105	134	158	189	210	232	257

In the experimental evaluation, we use five different GAs. In addition to the BRKGA, we have also included three GAs with permutational encoding and the GA already studied by Li et al. for the TCPP, in order to set an actual comparison basis. The details of the algorithms are:

- CX: It is a generational genetic algorithm with permutational encoding, roulette wheel selection, cycle crossover and swap mutation.
- OX: It is a generational genetic algorithm with permutational encoding, roulette wheel selection, order crossover and swap mutation.
- PMX: It is a generational genetic algorithm with permutational encoding, roulette wheel selection, partially-mapped crossover and swap mutation.
- Li et al.: It is the GA proposed by Li et al. [13].
- BRKGA: It is the BRKGA described in Section III-B.

The BRKGA parameter values used are 0.1 for the elite population fraction (p_e), 0.2 for the mutant population fraction (p_m), 0.5 for the elite allele inheritance probability (ρ_e) and $3 \times n$ for the population size, where n is the number of test cases of the instance.

For the rest of the GAs considered, the parameter settings are taken from the configuration used by Li et al. in [13], which consists in: 50 for the population size, 0.8 for the crossover probability and 0.1 for the mutation probability.

The stopping criterion used for the algorithms is to reach a maximum number of generations fixed a priori. In order to perform a fair comparison among the algorithms, the number of generations was determined for the BRKGA, and then the number of generations for the other algorithms was calculated guaranteeing that the number of solutions generated by every algorithm for each instance is exactly the same. The number of the generations for BRKGA is $6 \times n$.

All the algorithms are implemented in C++ and executed in a PC with a Quad Core Intel Xeon E5530 processor at 2.40 GHz with 48 GB RAM.

Since the algorithms used in the evaluation are stochastic, we use statistical tests to assess the significance of the experimental results obtained. Fifty independent runs for each algorithm and each instance have been executed and then the different metrics studied are computed. We analyze the statistical differences for the algorithms across the multiple

problems and instances using the Friedman's test [26]–[28]. The Friedman's test ranks the algorithms according to some particular metric. The test is used to check if the differences in the metric among the algorithms are statistically significant. Since more than two algorithms are involved in the study, a $1 \times N$ comparison using the Holm's post-hoc procedure is performed [26], [27]. All the statistical tests are performed with a confidence level of 95%.

B. Experimental Results

We analyze the numerical efficacy of the GAs studied in this work, especially the BRKGA. In particular, we address the following question: *is BRKGA able to provide better solutions than the GAs with permutational encoding and the GA proposed by Li et al. for the TCPP?* To answer this question, we consider two different metrics: the number of hits and the median of the numerical results.

The number of hits is computed as the total number of runs of each algorithm that are able to obtain the best APC value found for each instance. Table III presents the overall number of hits of each of the algorithms for each of the subject programs considered, while Table IV presents the mean Friedman's ranking according to the number of hits (the test is conducted using the 55 values, one for each instance) and the p-values adjusted with Holm's procedure.

TABLE III
TOTAL NUMBER OF HITS.

Subject	CX	OX	PMX	Li et al.	BRKGA
tcas	547	539	537	542	**550**
totinfo	512	498	526	544	**547**
schedule	**550**	**550**	**550**	**550**	**550**
schedule2	**550**	548	**550**	**550**	549
printtokens	111	54	87	171	**252**
printtokens2	224	92	156	213	**317**
replace	289	143	217	214	**365**

The best results are in bold.

TABLE IV
MEAN FRIEDMAN'S RANKING AND STAT. ASSESSMENT

Algorithm	Ranking	p_{value}	Stat. assessment
CX	2.8636	0.25e-1	✓
OX	3.8052	0.00e0	✓
PMX	3.3052	0.07e-3	✓
Li et al.	2.7987	0.25e-1	✓
BRKGA	2.2273	-	-

The number of hits show that BRKGA is the best performing algorithm (BRKGA has the best Friedman's ranking), as it is superior than the other algorithms in five out of seven programs and it has a numerical efficacy similar to the other GAs for the other two programs. The Friedman's test and the p-values show that BRKGA is significantly better than the rest of the GA considered in the experiment.

Let us now analyze the median of the numerical results [26]. We have considered $100 \times APC$ instead of the APC value for the sake of clarity. We have computed the medians of the results over the fifty runs for each algorithm and each instance of each subject program. It is impossible to include the complete results obtained for the 55 different instances (11 instances for each subject program) and for each algorithm due to its huge extension. For this reason, we summarize the results obtained using the average of the medians over the eleven instances of each program in Table V. Table VI presents the mean Friedman's ranking according to the medians of the numerical results and the p-values adjusted with Holm's procedure. It should be noted that the Friedman's test is conducted over the 55 values of the medians (one for each instance) for each algorithm.

TABLE V
AVERAGE OF THE MEDIANS OF THE NUMERICAL RESULTS.

Subject	CX	OX	PMX	Li et al.	BRKGA
tcas	95.119	95.119	95.119	95.119	95.119
totinfo	96.650	96.650	96.650	96.650	96.650
schedule	96.619	96.619	96.619	96.619	96.619
schedule2	96.758	96.758	96.758	96.758	96.758
printtokens	97.014	96.999	96.991	97.027	97.029
printtokens2	97.204	97.198	97.194	97.151	97.210
replace	97.060	97.058	97.051	97.048	97.063

TABLE VI
MEAN FRIEDMAN'S RANKING AND STAT. ASSESSMENT

Algorithm	Ranking	p_{value}	Stat. assessment
CX	2.8312	0.14e0	-
OX	3.1234	0.26e-1	✓
PMX	3.6558	0.01e-3	✓
Li et al.	2.9351	0.12e0	-
BRKGA	2.4545	-	-

The median of the numerical results also show that BRKGA is the best performing algorithm since BRKGA has the best Friedman's ranking. In particular, BRKGA is superior to the other four GAs in the three programs with a larger number of test goals and test cases, while in the other four programs are the algorithms are tied. This result suggest that BRKGA is better than the rest of the algorithms when large scenarios are considered. The Friedman's test and the p-values calculated show that BRKGA is significantly better than both OX and PMX. However, there is not enough statistical evidence that BRKGA is superior to CX and the GA proposed by Li et al.

From these numerical results, it is notorious that BRKGA outperforms the GAs with permutational encoding and OX and PMX operators. BRKGA has also better results than the permutational GA with CX operator and a GA specially conceived for tackling the TCPP. However, there is not enough statistical evidence to support this claim. As a consequence, we can conclude that BRKGA is at least as good as CX and Li et al. for the TCPP. This result is relevant since the GA proposed by Li et al. was specially conceived for solving the problem at stake.

Due to the large number of runs required for the experimental analysis, the execution platform was not used exclusively. For this reason, the computational performance is not reported. However, it can be pointed out that there was not a large difference in the runtime of the different algorithms considered, and that PMX is the only variant of the GAs evaluated with a shorter execution time than BRKGA.

V. Conclusions and Future Work

In this paper we have proposed a Biased Random-Key Genetic Algorithm for solving the test case prioritization problem. The experimental evaluation conducted on eleven instances of seven real-world programs, with up to 257 test cases and 208 test goals, showed that BRKGA is a very competitive algorithm for the problem. BRKGA is able to outperform two out of three GAs with permutational encoding. It is also able to obtain better numerical results (but without statistical significance) than the other permutational encoding based GAs and a GA proposed by Li et al. that was specially conceived for tackling the TCPP. It is also remarkable that BRKGA outperforms the other four GAs in the three programs with a larger number of test goals and test cases, i.e., in the larger instances considered.

From this work and the conclusions drawn, three main areas that deserve further study were identified. A first issue is to evaluate the BRKGA with other real-world benchmarks, specially including instances from test suites with a larger number of test goals and test cases than in this work. A second line of interest is the incorporation of specific knowledge to the search process through a local search operator or one of the specific heuristic analyzed by Li et al. This could lead to better solutions or may help to speed up the search process. And, finally, we aim to extend our analysis by solving similar problems from the field of software engineering, in order to continue evaluating the benefits of the Biased Random-Key Genetic Algorithm.

Acknowledgment

M. Pedemonte acknowledges support from PEDECIBA-Informática, Sistema Nacional de Investigadores and Comisión Sectorial de Investigación Científica, Uruguay.

References

[1] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, 2012.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed. Addison-Wesley Longman, 1989.

[3] C. Blum and A. Roli, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[4] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 1–61, 2012.

[5] F. Herrera, M. Lozano, and J. L. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis," *Artif. Intell. Rev.*, vol. 12, no. 4, pp. 265–319, Aug. 1998.

[6] D. Whitley and N. wook Yoo, "Modeling simple genetic algorithms for permutation problems," in *in Foundations of Genetic Algorithms*, 1995, pp. 163–184.

[7] J. F. Gonçalves and M. G. Resende, "Biased random-key genetic algorithms for combinatorial optimization," *Journal of Heuristics*, vol. 17, no. 5, pp. 487–525, 2011.

[8] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA journal on computing*, vol. 6, no. 2, pp. 154–160, 1994.

[9] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Softw. Test., Verif. Reliab.*, vol. 12, no. 4, pp. 219–249, 2002.

[10] M. Pedemonte, F. Luna, and E. Alba, "Systolic genetic search for software engineering: The test suite minimization case," in *Apps. of Evol. Comp. - Eur. Conf.*, ser. LNCS, vol. 8602, 2014, pp. 678–689.

[11] ——, "A systolic genetic search for reducing the execution cost of regression testing," *Applied Soft Computing*, vol. 49, pp. 1145 – 1161, 2016.

[12] A. B. Sánchez, S. Segura, and A. Ruiz-Cortés, "A comparison of test case prioritization criteria for software product lines," in *Proceedings of the 2014 IEEE International Conference on Software Testing, Verification, and Validation*, ser. ICST '14. IEEE Computer Society, 2014, pp. 41–50.

[13] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, 2007.

[14] Y.-C. Huang, K.-L. Peng, and C.-Y. Huang, "A history-based cost-cognizant test case prioritization technique in regression testing," *Journal of Systems and Software*, vol. 85, no. 3, pp. 626 – 637, 2012.

[15] W. Zhang, B. Wei, and H. Du, "Test case prioritization based on genetic algorithm and test-points coverage," *LNCS*, vol. 8630, no. PART 1, pp. 644–654, 2014.

[16] S. Sarwar, Y. Mahmood, Z. Qayyum, and I. Shafi, "Test case prioritization for nunit based test plans in agile environment," *LNCS*, vol. 8722, pp. 246–253, 2014.

[17] M. Ray and D. Mohapatra, "Multi-objective test prioritization via a genetic algorithm," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 261–270, 2014.

[18] D. Pradhan, S. Wang, S. Ali, T. Yue, and M. Liaaen, "Stipi: Using search to prioritize test cases based on multi-objectives derived from industrial practice," *LNCS*, vol. 9976, pp. 172–190, 2016.

[19] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "Hypervolume-based search for test case prioritization," *LNCS*, vol. 9275, pp. 157–172, 2015.

[20] D. E. Goldberg and R. Lingle Jr., "Alleles, loci, and the traveling salesman problem," in *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, J. J. Grefenstette, Ed. Lawrence Erlbaum Associates, Publishers, 1985.

[21] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'85, 1985, pp. 162–164.

[22] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A study of permutation crossover operators on the traveling salesman problem," in *Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application*, 1987, pp. 224–230.

[23] W. Spears and d. Jong, "On the virtues of parameterized uniform crossover," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 230–236.

[24] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proc. of the 16th Int. Conf. on Software engineering*, 1994, pp. 191–200.

[25] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Engg.*, vol. 10, no. 4, pp. 405–435, 2005.

[26] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[27] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 5th ed. Chapman and Hall/CRC, 2011.

[28] M. Pedemonte, F. Luna, and E. Alba, "A theoretical and empirical study of the trajectories of solutions on the grid of systolic genetic search," *Information Sciences*, vol. 445, pp. 97–117, 2018.