Test Case Prioritization and Reduction Using Hybrid Quantum-behaved Particle Swarm Optimization

Anu Bajaj
Machine Intelligence Research Labs (MIR Labs),
Auburn, Washington, USA
er.anubajaj@gmail.com, orcid.org/0000-0001-8563-6611

Ajith Abraham
Center for Artificial Intelligence, Innopolis University,,
Innopolis, Russia
Machine Intelligence Research Labs (MIR Labs),
Auburn, Washington, USA
ajith.abraham@ieee.org, orcid.org/0000-0002-0169-6738

2022 IEEE Congress on Evolutionary Computation (CEC) | 978-1-6654-6708-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/CEC55065.2022.9870238

Abstract—**Regression testing is an integral part of the software evolution and maintenance phase as it ensures that the modified software is working correctly after any upgrades. Test case prioritization and reduction minimize cost and effort needed for retesting by scheduling critical test cases before the less critical ones and removing redundant test cases. The criticality and redundancy of the test cases depend on several testing criteria. This paper empirically analyzed the effect of different testing criteria like code and fault coverage on the techniques' performance. This paper proposed a discrete Quantum-behaved particle swarm optimization (QPSO) for enhancing efficiency of test case prioritization. The algorithm is improved by replacing the random distribution with Gaussian probability to escape from the local optima. The evolution stagnation issue is further resolved by hybridizing it with genetic algorithm (QPSO-GA). In addition to prioritizing the test cases, the algorithm also reduces the test suite size through the test suite reduction approach. The experiments are conducted on different versions of three programs from the open-source software infrastructure repository. The performance is compared with the average percentage of statement coverage, fault detection, and their combinations with the cost. Consequently, suite reduction, fault detection capability losses, and coverage loss percentage are also drawn for test suite reduction. The proposed algorithms outperformed the random search, ant colony optimization, differential evolution, GA, PSO, and adaptive PSO for all the evaluation metrics.**

Index Terms—**regression testing, nature-inspired algorithms, test case prioritization, test suite reduction, particle swarm optimization, QPSO**

I. Introduction

The most challenging task of a software firm is to stay afloat in a competitive market by upgrading and maintaining the software to meet the changing demands. All software's test cases must be re-implemented to ensure that the quality is not compromised. It is known as regression testing [1]. Software gets more complicated with frequent updates, so the time and effort necessary for regression testing may rise. These bottleneck problems may be overcome by employing test case reduction, selection, and priority strategies that focus only on removing the redundant test cases, selecting the critical test case, and ranking the test cases based on predefined goals such as maximum code coverage, fault coverage, and requirements coverage.

With optimization techniques, the cost-effectiveness of regression testing may be further enhanced. The researchers attention was drawn to nature-inspired algorithms because of their simple structure and ease of application. The algorithms are developed by taking inspiration from the natural processes [2]. Swarm-intelligence and genetic algorithms of the biology-inspired class are the most often employed [3]. These algorithms have also been proven to be useful in regression testing [1]. For example, Li, Harman and Hierons [4] compared GA, hill climbing and greedy algorithms for Test Case Prioritization (TCP). It was observed that greedy methods worked better but Genetic Algorithm (GA) provided better fitness landscape.

Zhang *et al.* [5] used Ant Colony Optimization (ACO) to prioritize the test cases. The obtained results were better than GA, Particle Swarm Optimization (PSO) and Random Search (RS). A new adaptation strategy for permutation encoding was proposed to solve the TCP problem using Cuckoo Search Algorithm (CSA) [2]. Furthermore, Ahmed [6] used CSA to minimize the test suite for configuration-aware software testing. Few researchers have used PSO like Khatibsyarbini, Isa, and Jawawi [7] implemented PSO using string distances for ordering the test cases. The proposed algorithm performed better than the nearest neighbor and RS for the real-world TSL dataset. Binary constraint PSO and its hybrid versions with local search techniques were developed to select the test cases based on the redundancy and the required efforts [8].

Standard PSO algorithm have issues, e.g., stuck into local optima and premature convergence [9]. Observations suggested that improved and hybrid versions of PSO outperformed the standard PSO to test the complex programs [8]. One such improvement is the Quantum-behaved PSO (QPSO) algorithm. It is inspired by the quantum behavior of particles, i.e., particles can move in a wide search space for global convergence [11]. The algorithm shows promising results in various applications [10]- [13]. On the other side, it has not been examined in TCP. We introduced discrete QPSO that perturbs the population using asexual genetic operator [2]. However, it also has some disadvantages like premature convergence. Therefore, we have hybridized it with GA

This research has been financially supported by The Analytical Center for the Government of the Russian Federation (Agreement No. 70-2021-00143 dd. 01.11.2021, IGK000000D730321P5Q0002).

to accelerate the performance in the last iterations, and to avoid premature convergence uniform distribution of quantum-behaved particles is replaced with the Gaussian distribution [10]. Besides this, we have also improved the method for reducing redundancy. The main contributions of this paper are:

- Discretized QPSO algorithm to solve the combinatorial TCP problem using asexual reproduction algorithm based fix-up mechanism.
- Suggested a hybrid algorithm using advantages of particle swarm and genetic algorithm, i.e., hybrid QPSO-GA.
- Included the Test Suite Reduction (TSR) method in the TCP to remove redundant test cases.
- Verified algorithm's robustness against testing goals like code coverage, fault coverage and cost reduction.
- Compared proposed work against baseline approach: Random search (RS) and different state-of-the-art algorithms like ACO, Differential Evolution (DE), GA, PSO, and adaptive PSO (APSO).

II. PSO and Quantum-Behaved PSO

A. Particle Swarm Optimization (PSO)

PSO is influenced by the flocking, swarming and herding behavior of agents called particles. The particles change their flights using self and neighbor's flying experience. Each particle with its self-experience knows the location of food known as personal best position (P). At the same time, the particle is aware of the swarms' best found location, i.e., global best position (G). This phenomena is imitated to solve the real world problems. In other words, the swarm consists of particles that move randomly in the search space with velocity v_i at location x_i and change their positions using the self experience, social and cognitive behavior [14]. Mathematically, the location and the velocity of each particle i at g^{th} generation are formulated as:

$$v_i^{g+1} = wv_i + c_1 r_1 \left(P_i^g - {x_i}^g\right) + c_2 r_2 \left(G^g - {x_i}^g\right) \quad (1)$$

$$x_i^{g+1} = x_i^g + v_i^{g+1} \quad (2)$$

here w is the inertia weight for controlling the effect of prior velocity; c_1 and c_2 are the constants that controls the attraction rates of these social and cognitive components; and r_1 and r_2 are the uniform random numbers between $[0,1]$.

B. Quantum Behaved PSO

PSO cannot guarantee the global convergence [12] so a more robust version of PSO is developed called QPSO [9]. It finds the path of the quantum-behaved particles by assuming that the N particles with δ potential and specified energy are well-centered in each dimension of n-dimensional Hilbert search space. So, according to Monte Carlo method, j^{th} component of the particle's position at g^{th} generation:

$${x_{ij}}^{g+1} = {a_{ij}}^g \pm \frac{L_{ij}^g}{2} \ln\left(\frac{1}{u_{ij}^g}\right) \quad (3)$$

$$L_{ij}^g = 2\theta|mean_j^g - x_{ij}^g| \; and \; mean_j^g = \frac{1}{N}\sum_i^N x_{ij}^g \quad (4)$$

Where u_{ij}^g is a uniform random value ranging from 0 to 1, θ is contraction–expansion coefficient and a_{ij}^g is particle's local attractor. So particle's position in the QPSO algorithm can be obtained as:

$$x_i^{g+1} = \begin{cases} a_{ij}^g - \theta|Mbest_j^g - x_{ij}^g|\ln(1/u_{ij}^g) & : r(0,1) > 0.5 \\ a_{ij}^g + \theta|Mbest_j^g - x_{ij}^g|\ln(1/u_{ij}^g) & : otherwise \end{cases} \quad (5)$$

The particles move around the local attractor a_{ij}^g in each generation and it is formulated with the P and G best positions as follows:

$$a_{ij}^g = \phi_{ij}^g P_{ij}^g + (1 - \phi_{ij}^t)G_j^g, \; \phi_{ij}^g \sim (0,1) \quad (6)$$

Whereas, the position distribution of the particles for the next generation is calculated with the mean $Mbest$ of the P best positions of the particles.

$$Mbest_j^g = \frac{1}{N}\sum_i^N P_{ij}^g \quad (7)$$

So, the underlying difference between the PSO and QPSO lies in two ways: 1) wide search space due to exponential distribution of the particles. 2) the distance between the particle and its companions are taken into account whereas, in PSO the particles move freely to converge to global best. Another advantage is it has only one parameter, i.e., θ that need to be controlled for convergence and its value is decreased linearly:

$$\theta = (\theta_{max} - \theta_{min}) * (nGen - g)/nGen + \theta_{min} \quad (8)$$

As it is easy to implement and has been implemented on various applications [12]. Therefore, in this paper, we have attempted to apply QPSO algorithm for discrete optimization problem and compare its performance with the state-of-the-art algorithms.

III. Proposed Discretized Hybrid QPSO-GA

This section describes the proposed hybrid QPSO-GA algorithm with respect to TCP. It is done in three subsequent steps. Firstly, it updates the population with the proposed asexual reproduction operator (ARO). Secondly, Gaussian probability is used to avoid premature convergence. Thirdly, the stagnation issue is resolved with GA swap mutation operator and finally the TSR approach is followed by the TCP algorithm for reducing the redundancy as explained:

A. Population Update

One of the key issues in the successful application of any nature-inspired algorithms is mapping the problem with the algorithm, as it directly affects the performance and feasibility. The PSO and QPSO algorithms were designed for continuous problems. The original approach cannot be used directly for a discrete combinatorial issue. Therefore, we have chosen permutation encoding to represent the solution since proper mapping improves the algorithm's speed and efficacy. We modified the real values to permutation sequences by using the asexual reproduction algorithm [2].

Algorithm 1 describes the working of fix-up mechanism where the current solution inherit the parent solution's features by creating a link between actual numbers and test case series. It retains the offspring's feasible values during development of the bud from parent (larva). To put it another way, the algorithm refreshes the results by replacing the duplicate and out of bound particles with don't care values (*). To produce a proper solution, these infeasible values are substituted with values from the prior solution. For instance, when x= [4, 6, 5, 2, 1, 3] is updated to y= [6.2, 7.4, 2.4, 5.7, 2.3, 1.1], y is adjusted to [6, 7, 2, 5, 2, 1], yielding [6, *, 2, 5, *, 1]. The correct solution is developed by acquiring the remaining particles (genetic characteristics) of new offspring from the the prior solution (parent) as [6, 4, 2, 5, 3, 1].

Algorithm 1 Asexual Reproduction Fix-Up Mechanism

```
1: $x_p$ =$x_i(g)$
2: for $i = 1, 2, \ldots, nPop$ do
3:     $x_i(g+1) = round(x_i(g+1))$
4:     if $x_i(g+1) < 1$ or $x_i(g+1) > n$ then
5:         $x_i(g+1)$=$^*(don't\ care)$
6:     end if
7: end for
8: $y_l$=$x_i(g+1)$
9: $\Delta$=$setdiff(1 : length(y_l), unique(y_l))$
10: $x_b$=$replace(x_p$,$\Delta_{id}$,$\Delta)$
11: $x_i(g+1)$=$x_b$
12: Return: $x_i(g+1)$
```

B. Gaussian Probability

The uniform random number u of (5) is replaced by $Gu(x)$ (9), i.e., with the absolute value of Gaussian distribution $N(0, 1)$ to avoid premature convergence.

$$Gu(x) = \frac{2}{\sqrt{2\pi}} e^{-x^2/2}, x \geq 0 \tag{9}$$

Therefore, (5) is updated as

$$x_i^{g+1} = \begin{cases} a_{ij}^g - \theta|Mbest_j^g - x_{ij}^g| \ln(1/Gu_{ij}^g) & : r(0,1) > 0.5 \\ a_{ij}^g + \theta|Mbest_j^g - x_{ij}^g| \ln(1/Gu_{ij}^g) & : otherwise \end{cases} \tag{10}$$

C. Diversity Enhancement

Though QPSO algorithm provides the wide search space to the particles throughout the generations. However, the distance between the particles distribution $|Mbest_j^g - x_{ij}^g|$ narrows down in the final phase of search. It leads to loss of exploration and stagnates the evolution process if the solution does not improve for successive δ generations. It requires external interference to progress further. So, GA's swap mutation operator is used to escape from the local optima problem. It allows the particles to move away from the current point, hence, reduces the interference and improves the global search. On the other hand, it may create a fresh solution that differs from the prior one. There's a risk that it can overlook the best, high-quality solution. As a result, in the proposed method, a decision criterion about when to apply the mutation operator is made to avoid this risk. The following condition is tested before each generation begins:

$$\sum_{i=g}^{g-10} gfit\,(i) - gfit\,(i-1) \leq \delta \tag{11}$$

Here δ is the threshold for finding the disturbance in the solutions. In other words, it accelerates the search when consecutive ten generations' global fitness $(gfit)$ values fall in a small limit.

D. Test Suite Reduction (TSR)

Each iteration's current best solution is executed for removing the faults/statements redundancy by selecting the initial m test cases to reduce suite cost and size. This method has the benefit of revealing how exactly the test cases are prioritised. The greater the prioritising, the fewer test cases are required to meet the complete coverage criterion. The pseudo-codes of TSR and hybrid QPSO-GA are presented in Algorithms 2 and 3 .

Algorithm 2 Test Suite Reduction Algorithm

```
1: Define test fault matrix TFM, Prioritized test array PT and
   Faults position array FP
2: Initialize reduced array of test cases indices RSInd[FP]=0
3: Index=find(TFM(1,FP)=1)
4: Fill the RSInd with the faults positions covered by the first
   test case in PT
5: RSInd(Index)=1
6: Find other test cases needed for full coverage
7: for $t = 1, 2, \ldots, size(PT)$ do
8:     for $f = 1, 2, \ldots, size(FP)$ do
9:         if (TFM(PT(t),FP(f))=1 and RSInd(f)=0)) then
10:            RSInd(f)=t
11:        end if
12:    end for
13: end for
14: Return: Minimized Test Suite=PT(RSInd)
```

IV. Experimental Setup

This Section outlines an empirical study, including research questions, datasets, evaluation metrics, and the algorithms with which the proposed algorithm is compared. The formulated research questions are:

RQ1. How the proposed algorithm performed for TCP? The goal is to see whether the proposed algorithm outperformed other algorithms. It also determines which algorithm obtains the best results and the impact of various testing conditions on the algorithms' performance.

RQ2. How the proposed algorithm performed for TSR? The purpose is to compare the proposed algorithm's performance to that of the other algorithms. Additionally, to determine which testing criteria optimise TSR. In addition, to investigate how it affects the test suite's coverage, fault detection capacity and cost reduction.

Algorithm 3 QPSO-GA Algorithm

```
1: Define $nPop$, $nGen$, $\theta_{max}$, $\theta_{min}$ and $\delta$
2: Initialize random population $x_i$
3: for $g = 1, 2, \ldots, nGen$ do
4:     for $i = 1, 2, \ldots, nPop$ do
5:         Calculate fitness $f(x_i)$
6:         Update $P_i$ and $G$ solutions
7:         Update $x_i(g+1)$ using $(10)$
8:         Repair the solution using Algorithm 1
9:     end for
10:    if $G$ does not improve for $\delta$ attempts then
11:        Create new solution using swap mutation
12:        $x_i(g+1)$=$x_i(g)$+$x_\mu$
13:    end if
14:    Apply test suite reduction Algorithm 2
15: end for
16: Return: Final solution
```

A. Experimental design

The algorithms used for comparison purposes are RS, GA, PSO, ACO, DE, APSO. These algorithms are implemented in MATLAB R2017 installed on an Dell laptop having Intel i5 processor, Windows 11, and 8GB RAM. The algorithms are executed for 30 runs because of their stochastic behavior. These are applied on several versions of *jtopas, ant* and *jmeter* programs of software infrastructure repository (SIR) [15] (see Table I).

TABLE I
SUBJECT PROGRAMS

Programs	Versions	KLOC	Classes	Methods	Test Cases	Type
ant	7	80.4	650	7524	878	JUnit
jtopas	4	5.4	50	748	209	JUnit
jmeter	5	43.4	389	3613	97	JUnit

Parameter settings play a major role in the performance of the algorithms [16]. So, we carefully choose the parameters from the literature followed by Taguchi method for appropriate values (see Table II).

TABLE II
PARAMETER SETTINGS OF THE ALGORITHMS

Algorithms	Parameters values
GA	$p_{cr} = 0.8$, $p_m = 0.1$, tournament selection, ordered crossover
PSO	$c_1 = 1.5$, $c_2 = 2$, $w_{min} = 0.4$, $w_{max} = 0.8$
ACO	$\alpha = 0.3$, $\beta = 0.9$, $init = 20$, $\rho = 0.2$
DE	$pc = 0.6$, $F = 0.8$
QPSO	$\theta_{min} = 0.4$, $\theta_{max} = 1$
QPSO-GA	$\theta_{min} = 0.3$, $\theta_{min} = 1$, $\delta = 10$
Common Parameters	$nPop = 100$, $nGen = 1000$

B. Performance Measures

The following performance measures are used to validate the efficiency and efficacy of these algorithms:

1) Test Case Prioritization: To assess the robustness of the proposed technique, the test cases are selected using statement and fault coverage criteria. As a result, commonly used fitness measurements and effectiveness measures are defined as follows:

Average Percentage of Fault Detection (APFD) is a measure of how well a system detects faults. It finds a weighted average of the detected defects based on where they are in the test suite [17]. It's computed as follows:

$$APFD = 1 - \frac{\sum_{i=1}^{m} TF\left(i\right)}{n * m} + \frac{1}{2 * n} \tag{12}$$

The location of the test case that detects the i^{th} fault is denoted by $TF(i)$, and the faults covered by n test cases is denoted by m. It's value lies in between 0 and 100, with greater being better.

Average Percentage of Fault Detection with Cost (APFDc): APFD is based on the assumption of consistent test case costs and fault severity levels, which is rarely the case. As a result, a cost-conscious measure, APFDc, has been developed [18], which incorporates various costs and fault severity levels in APFD and is written as:

$$APFDc = 1 - \frac{\sum_{i=1}^{m} fs\left(i\right) * \left(\sum_{j=TF(i)}^{n} cost\left(j\right) - \frac{1}{2} cost\left(TF\left(i\right)\right)\right)}{\sum_{i=1}^{n} cost\left(i\right) * \sum_{i}^{m} fs\left(i\right)} \tag{13}$$

cost $(TF(i))$ is the cost of the test case that discovers the i^{th} fault first having fault severity of $fs(i)$ and $cost(j)$ is test execution cost of j^{th} test case. The Average Percentage of Statement Coverage (APSC) and APSC with cost (APSCc) are calculated in the same way as the APFD and APFDc.

2) Test Suite Reduction: Test suite reduction percentage and cost reduction percentage are two regularly utilised effectiveness indicators. The test suite reduction, which comes after the TCP, decreases suite size by employing 100% statement/fault coverage.

Test Reduction Percentage (TRP): It's the percentage reduction in the size of the test suite.

$$TRP = \frac{n - s}{n} * 100 \tag{14}$$

Here s indicates the test cases selected from n test cases.

Coverage Loss Percentage (CLP): It's the proportion of statements left undetected slu by reduced test suite to total statements covered tsc.

$$CLP = \frac{slu}{tsc} * 100 \tag{15}$$

Fault Detection Capability Loss Percentage (FLP): Ratio of faults not covered by reduced test suite nfl to total faults covered tfc:

$$FLP = \frac{nfl}{tfc} * 100 \tag{16}$$

Cost Reduction Percentage (CRP): This is the percentage of the test suite's cost that is reduced $rcost$ when compared to the original suite's cost $tcost$.

$$CRP = \frac{rcost}{tcost} * 100 \tag{17}$$

V. Results and Analysis

The proposed algorithms are empirically evaluated using various fitness functions. The impact of TSR on fault/statement coverage loss and reduction in cost is also investigated. Cumulative average of all the versions of a program is used to calculate its experimental outcomes. The mean of 30 runs is used to calculate the performance metrics for each version. The performance of the algorithms is statistically compared using a one-way ANOVA test having a p-value of 0.05. The null hypothesis is rejected if the p-value is less than 0.05, indicating that the algorithms' means are different. It is followed by a Tukey simultaneous test to assess the pairwise comparison of the algorithms. Furthermore, boxplots and interval plots for fitness measurements and test reduction percentages are demonstrated.

A. Performance analysis of TCP (RQ 1)

Table III shows the mean fitness values of the performance metrics, as well as their corresponding Tukey group ranks, for all the programs. Observations show that all the nature-inspired algorithms are statistically different from RS, with a p-value less than 0.05, for all testing criteria. Moreover, it suggests that there is an insignificant difference between means of 1) RS and ACO 2) GA and PSO 3) DE and GA in most of the cases. APSO is significantly better than DE for APSCc and one out of three programs for other criteria. There is insignificant difference between QPSO and APSO for statement coverage criteria. On the other hand, QPSO-GA is better than QPSO except APSCc, however it has higher mean values, which makes QPSO-GA superior to all other algorithms.

The box plots, as illustrated in Figure 1, graphically display the fitness values' distribution. It shows that the proposed QPSO-GA algorithm possesses the high-quality solutions comparatively. It is also discovered that variance for fault coverage is more than statement coverage because of faults' spread throughout software. In other words, most test cases cover almost same statements, resulting into squeezed boxplots compared to fault coverage. On the other side, the variance in fault coverage with cost is the biggest. It may be claimed that, apart from the vast fault distribution, these variances are due to the varied execution costs that result in different test case ordering. Therefore, test case execution cost is also important criterion. Because if two test cases cover the same amount of faults but have different costs, the algorithm will prioritise the test case with the lowest cost above the other.

B. Performance analysis of TSR (RQ 2)

Table IV shows the average test reduction percentages of all algorithms, as well as their Tukey group ranks, for all the programs. Observations suggest that though the mean values of all the algorithms are different however most pairs are statistically insignificant like 1) RS and ACO, 2) GA and PSO 3) GA and DE 4) DE and APSO. QPSO and QPSO-GA performed similar in case of fault coverage criteria while QPSO-GA is superior to QPSO for statement criteria. It depicts that ACO and DE

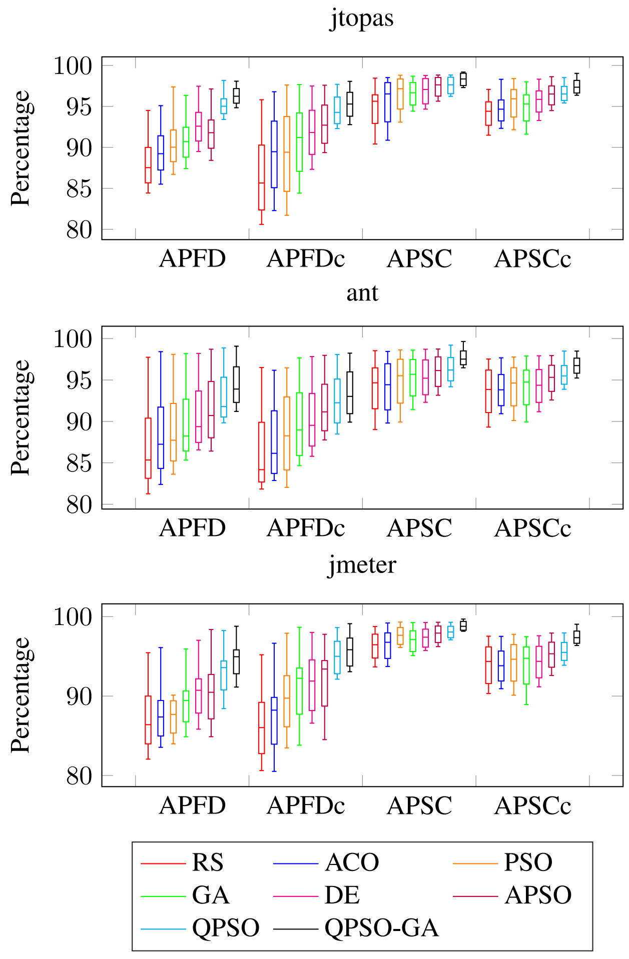

Fig. 1. Boxplots of algorithms for TCP of different testing criteria

have less significant suite size compared to APSO, QPSO, and QPSO-GA algorithms showing the inefficiency of the ACO and DE compared to the PSO variants. Overall, the proposed algorithm QPSO-GA works better than all the algorithms in each case.

Interval plots are used to depict the distribution of test reduction percentages (see Figure 2). For all of the testing criteria, the proposed algorithms QPSO and QPSO-GA outperformed RS, GA, PSO, ACO, and DE algorithms. As observed from Table IV and Figure 2, TSR is larger fir statement coverage than fault coverage. It is because of many redundant statements compared to faults that spread over the program. TSR also observes underlying difference in the TCP methods' performances, i.e., if the algorithm correctly prioritizes the test cases. Alternatively, it might be due to chance effect or program characteristics if an algorithm performs well for TCP but can not decrease the test suite for 100 percent coverage requirements.

Table V shows that QPSO-GA has least loss in the coverage as compared to other algorithms for all the testing criteria. However, PSO has least coverage loss for APFD and APFDc of *jmeter* comparatively. Observations also depict that the

TABLE III
FITNESS COMPARISON OF ALGORITHMS FOR TCP

Programs	Algorithms	Fitness values and Tukey Group Ranking (%)							
		APFD	TR	APFDc	TR	APSC	TR	APSCc	TR
jtopas	RS	88.132	F	87.176	E	95.816	E	94.908	E
	ACO	89.480	E	91.066	D	97.687	D	95.174	D
	PSO	90.236	D	91.274	CD	98.046	BC	96.601	C
	GA	91.204	D	92.664	BC	97.417	C	95.759	D
	DE	92.217	C	92.672	B	98.168	BC	96.365	C
	APSO	93.104	C	93.787	B	98.383	B	97.125	B
	QPSO	95.217	B	95.059	A	98.401	B	96.999	B
	QPSO-GA	96.535	A	95.754	A	98.904	A	97.778	A
ant	RS	85.684	G	84.800	E	95.274	E	94.884	D
	ACO	88.226	F	87.724	D	95.871	E	94.755	D
	PSO	88.627	EF	90.264	C	96.535	CD	95.606	C
	GA	88.934	DE	90.863	C	96.621	CD	95.404	C
	DE	90.377	CD	90.786	C	96.294	D	95.306	C
	APSO	91.789	BC	92.333	B	96.988	BC	95.972	B
	QPSO	92.365	B	93.332	AB	96.799	B	95.884	B
	QPSO-GA	94.429	A	94.081	A	97.830	A	97.080	A
jmeter	RS	86.895	F	87.168	E	96.594	F	97.007	E
	ACO	88.334	E	89.081	D	98.166	E	97.789	E
	PSO	88.689	E	90.677	C	98.711	BCD	98.391	BC
	GA	90.256	D	92.846	BC	98.438	D	98.084	DE
	DE	91.573	C	93.999	B	98.651	CD	98.229	CD
	APSO	91.563	C	93.841	B	98.818	BC	98.593	B
	QPSO	94.023	B	96.509	A	98.888	B	98.511	B
	QPSO-GA	95.407	A	97.155	A	99.471	A	99.138	A

TABLE IV
ALGORITHMS' COMPARISONS FOR TSR

Programs	Algorithms	Fitness functions wise TSP and Tukey Group Ranking (%)							
		TSP_{APFD}	TR	TSP_{APFDc}	TR	TSP_{APSC}	TR	TSP_{APSCc}	TR
jtopas	RS	81.917	E	81.867	D	81.150	D	82.017	E
	ACO	83.008	DE	83.433	C	83.883	CD	82.742	E
	PSO	84.317	CD	84.300	C	86.575	BC	85.942	CD
	GA	84.633	C	84.850	BC	84.800	BC	83.825	DE
	DE	84.467	CD	84.308	C	84.892	BC	84.225	CDE
	APSO	86.167	B	86.100	B	86.958	B	86.858	BC
	QPSO	85.558	B	86.950	A	87.375	B	87.858	B
	QPSO-GA	**87.200**	A	**87.167**	A	**89.142**	A	**88.867**	A
ant	RS	74.200	D	73.790	D	75.705	D	76.090	D
	ACO	76.705	CD	75.852	D	78.043	CD	77.000	CD
	PSO	76.919	CD	77.271	C	80.067	BC	79.124	BC
	GA	78.067	B	77.857	C	79.995	BC	78.943	BC
	DE	80.019	B	79.129	B	81.095	BC	80.019	B
	APSO	80.205	B	80.043	B	81.443	B	81.267	B
	QPSO	81.038	A	80.881	B	81.919	B	81.086	B
	QPSO-GA	**81.352**	A	**81.224**	A	**83.719**	A	**83.524**	A
jmeter	RS	77.883	D	78.100	E	88.250	D	87.933	D
	ACO	78.967	CD	79.267	D	88.923	D	88.707	D
	PSO	78.930	CD	80.523	CD	91.083	B	90.740	C
	GA	79.820	BCD	80.203	CD	89.947	C	89.600	C
	DE	82.093	B	82.380	B	91.733	B	91.060	B
	APSO	80.917	B	81.730	C	91.580	B	90.967	B
	QPSO	83.467	A	83.957	A	91.833	B	91.590	B
	QPSO-GA	**83.423**	A	**83.983**	A	92.517	A	**92.047**	A

TABLE V
CLP, FLP AND CRP OF THE ALGORITHMS FOR TSR

Program	Algorithms	CLP, FLP and CRP of TSR							
		CLP_{APFD}	CLP_{APFDc}	FLP_{APSC}	FLP_{APSCc}	CRP_{APFD}	CRP_{APFDc}	CRP_{APSC}	CRP_{APSCc}
jtopas	RS	11.55	12.27	28.36	26.94	83.66	83.91	86.95	86.11
	ACO	9.90	10.11	25.38	23.35	84.65	85.41	84.85	87.29
	PSO	11.00	9.85	23.69	23.18	82.83	82.94	84.05	83.38
	GA	12.33	14.33	**23.35**	**20.83**	81.59	81.64	82.02	82.16
	DE	12.57	13.52	25.74	25.70	84.06	84.38	86.31	84.17
	APSO	10.28	10.50	30.47	30.20	84.63	85.04	87.95	86.93
	QPSO	8.96	9.58	29.91	29.58	85.01	85.68	88.57	87.19
	QPSO-GA	**7.15**	**7.32**	28.56	27.50	**88.28**	**87.50**	**89.01**	**88.57**
ant	RS	7.01	7.14	18.22	17.59	75.88	76.86	79.98	78.90
	ACO	6.33	6.43	18.71	16.37	77.56	77.67	79.79	80.96
	PSO	5.79	5.83	16.62	**15.24**	75.59	75.13	77.82	76.99
	GA	6.63	6.67	**16.17**	16.65	73.42	73.07	75.80	75.71
	DE	6.99	6.95	19.13	17.44	76.95	77.42	79.80	78.30
	APSO	6.38	6.13	19.79	19.27	77.46	77.79	81.09	79.87
	QPSO	5.56	5.89	20.34	19.34	78.56	78.44	81.67	80.23
	QPSO-GA	**4.75**	**5.33**	19.36	18.83	**80.98**	**79.90**	**82.06**	**81.58**
jmeter	RS	**4.05**	4.25	30.57	29.95	79.61	80.42	90.74	90.48
	ACO	4.16	4.13	26.37	**24.32**	80.77	80.80	89.54	91.27
	PSO	3.87	3.82	27.11	25.36	78.72	78.77	88.54	88.27
	GA	4.17	4.24	**27.01**	26.84	78.35	77.76	88.17	87.18
	DE	4.15	4.53	28.16	27.37	80.11	79.75	90.49	88.99
	APSO	3.83	4.23	30.70	30.60	80.75	81.38	91.39	90.80
	QPSO	3.95	3.87	29.50	28.56	81.28	81.37	91.67	90.50
	QPSO-GA	4.29	**3.43**	27.92	27.50	**81.91**	**81.44**	**91.97**	**91.91**

FLP of APSC and APSCc is comparatively higher to CLP of APFD and APFDc. The reason for this is larger test suite redundancy in statement coverage, therefore, reduction of test suite according to fault coverage, may lead to less coverage loss. The RS and ACO algorithms have higher fault coverage loss than other algorithms. It may be deduced that the loss of coverage and the reduction in test suite size are inversely proportionate. Another thing to keep in mind, if we lower the test suite size based on one criterion, we may lose a proportion of other metrics. It might happen as a result of the significant reduction in the size of the test suite. Table V clearly shows that the CRP of QPSO-GA and QPSO is larger than the other algorithms. It is also observed that the cost reduction is proportionate to the TSR, i.e., larger the decrease in size of suite, lower will be the test cost.

Overall, QPSO and QPSO-GA surpass the ACO, GA, DE and PSO for convergence speed, robustness, and quality of final solutions. Though quality of solutions of QPSO-GA is better than QPSO, but the only disadvantage of QPSO-GA is that it requires one more parameter to fine-tune, i.e., δ. It can be adjusted by adaptive/dynamic parameter settings. However, at the expense of one extra-parameter, QPSO-GA have demonstrated superior search capabilities to solve the TCP and TSR in all the three subject programs.

VI. CONCLUSIONS

We suggested the discretized QPSO algorithms for test case prioritization and reduction, and compared them to current nature-inspired methods. Empirical results show that the suggested algorithms outperformed these algorithms for all performance criteria. The QPSO-GA's advantage for TCP was validated by a statistical test. Furthermore, the proposed algorithms' usefulness was shown using boxplots and interval plots. In TSR, the QPSO-GA and QPSO performed similarly for fault coverage. The suggested algorithms, on the other hand, performed better than the APSO in terms of statement coverage and cost reduction % when it came to decreasing the test suite. The proposal of a test case selection approach that picks a reasonable amount of test cases without sacrificing software quality is one of our future work. We also plan to investigate alternative variants of QPSO to validate on more real-world applications for generalization.

REFERENCES

[1] Yoo, S. and Harman, M., "Regression testing minimization, selection and prioritization: a survey," Software Testing, Verification and Reliability, vol. 22. no. 2, 2012, pp.67-120.

[2] Bajaj, A. and Sangwan, O.P., "Discrete cuckoo search algorithms for test case prioritization," Applied Soft Computing, vol. 110, 2021, p.107584.

[3] Fister, J.I., Yang, X.S., Fister, I., Brest, J., and Fister, D., A brief review of nature-inspired algorithms for optimization," arXiv preprint arXiv:1307.4186, 2013, pp. 116-122.

[4] Li, Z., Harman, M. and Hierons, R.M., "Search algorithms for regression test case prioritization," IEEE Transactions on Software Engineering, vol. 33, no. 4, 2007, pp.225-237.

[5] Zhang, W., Qi, Y., Zhang, X., Wei, B., Zhang, M. and Dou, Z., "On test case prioritization using ant colony optimization algorithm." In 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2019, pp. 2767-2773.

[6] Ahmed, B.S., "Test case minimization approach using fault detection and combinatorial optimization techniques for configuration-aware structural testing", Engineering Science and Technology, an International Journal, vol. 19, no. 2, 2016, pp.737-753.

[7] Khatibsyarbini, M., Isa, M.A. and Jawawi, D.N.A., "Particle swarm optimization for test case prioritization using string distance," Advanced Science Letters, vol. 24, no. 10, 2018, pp.7221-7226.

[8] De Souza, L.S., Prudêncio, R.B., Barros, F.D.A. and Aranha, E.H.D.S., "Search based constrained test case selection using execution effort," Expert Systems with Applications, vol. 40, no. 12, 2013, pp.4887-4896.

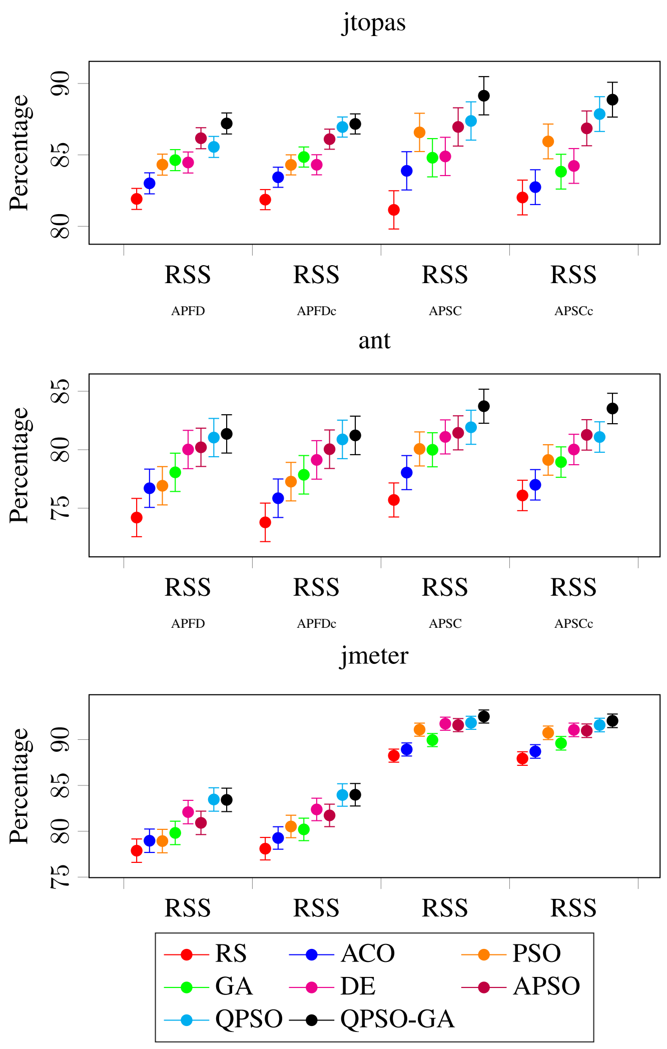

Fig. 2. Interval plots of algorithms for TSR of different testing criteria

[9] Sun, J., Xu, W. and Feng, B., ”A global search strategy of quantum-behaved particle swarm optimization,” In IEEE Conference on Cybernetics and Intelligent Systems, vol. 1, 2004, pp. 111-116.
[10] Coelho, L. dos S., ”Gaussian quantum-behaved particle swarm optimization approaches for constrained engineering design problems,” Expert Systems with Applications, vol. 37, no. 2, 2010, 1676–1683.
[11] Lukemire, J., Mandal, A. and Wong, W.K., ”d-qpso: a quantum-behaved particle swarm technique for finding d-optimal designs with discrete and continuous factors and a binary response,” Technometrics, vol. 61, no. 1, 2019, pp.77-87.
[12] Omkar, S.N., Khandelwal, R., Ananth, T.V.S., Naik, G.N. and Gopalakrishnan, S., ”Quantum behaved particle swarm optimization (QPSO) for multi-objective design optimization of composite structures,” Expert Systems with Applications, vol. 36, no. 8, 2009, pp.11312-11322.
[13] Fang, W., Sun, J. and Xu, W., ”A new mutated quantum-behaved particle swarm optimizer for digital IIR filter design,” EURASIP Journal on Advances in Signal Processing, 2009(2010), pp.1-7.
[14] Kennedy, J. and Eberhart, R., ”Particle swarm optimization,” In Proceedings of ICNN'95-international conference on neural networks, vol. 4, 1995, pp. 1942-1948. IEEE.
[15] Do, H., Mirarab, S., Tahvildari, L. and Rothermel, G., ”The effects of time constraints on test case prioritization: A series of controlled experiments,” IEEE Transactions on Software Engineering, vol. 36, no. 5, 2010, pp.593-617.
[16] Bajaj, A. and Sangwan, O.P., “Study the impact of parameter settings and operators role for genetic algorithm based test case prioritization”, In Proceedings of International Conference on Sustainable Computing in Science, Technology and Management, http://dx.doi.org/10.2139/ssrn.3356318, 2019, pp. 1564-1569.
[17] Elbaum, S., Malishevsky, A.G., and Rothermel, G., ”Test case prioritization: A family of empirical studies,” IEEE Transactions on Software Engineering, vol. 28, no. 2, 2002, pp.159-182.
[18] Malishevsky, A.G., Ruthruff, J.R., Rothermel, G. and Elbaum, S., ”Cost-cognizant test case prioritization,” Technical Report TR-UNL-CSE-2006-0004, University of Nebraska-Lincoln, 2006, pp. 97-106.