Quantum-Inspired Genetic Algorithm for Solving the Test Suite Minimization Problem

HAGER HUSSEIN[1,2], AHMED YOUNES[3,4], WALID ABDELMOEZ[1,5]
[1]Department of Software Engineering, College of Computing and Information Technology
Arab Academy for Science and Technology
Alexandria
EGYPT
[2]hager.hussein@gmail.com
[3]Department of Mathematics and Computer Science, Faculty of Science
Alexandria University
Alexandria
EGYPT
[4]School of Computer Science
University of Birmingham
Birmingham, B15 2TT
UNITED KINGDOM
ayounes2@yahoo.com
[5] walid.abdelmoez@gmail.com

Abstract: - Test Suite Minimization problem is a nondeterministic polynomial time (NP) complete problem in software engineering that has a special importance in software testing. In this problem, a subset with a minimal size that contains a number of test cases that cover all the test requirements should be found. A brute-force approach to solving this problem is to assume a size for the minimal subset and then search to find if there is a subset of test cases with the assumed size that solves the problem. If not, the assumed minimal size is gradually incremented, and the search is repeated. In this paper, a quantum-inspired genetic algorithm (QIGA) will be proposed to solve this problem. In it, quantum superposition, quantum rotation and quantum measurement will be used in an evolutionary algorithm. The paper will show that the adopted quantum techniques can speed up the convergence of the classical genetic algorithm. The proposed method has an advantage in that it reduces the assumed minimal number of test cases using quantum measurements, which makes it able to discover the minimal number of test cases without any prior assumptions.

Key-Words: - Quantum-Inspired Genetic Algorithm, Classical Genetic Algorithm, Minimization Problem, Software Testing, Optimization

Received: January 26, 2020. Revised: July 17, 2020. Accepted: August 10, 2020. Published: August 17, 2020.

1 Introduction

Software testing is a widely studied approach for assessing and improving software quality. Various techniques exist to perform software testing [1]. For example, there are functional techniques and structural techniques. The functional techniques are when the program is viewed as a black box and the test cases selection is based on the requirement or the software design specification [1]; meanwhile, the structural techniques view the software as a white box and select the test cases based on the software's implementation[1]. Search-based software engineering (SBSE) is used to solve optimization problems. When there is a space of solutions, SBSE can be used to find the best solution among the candidate solutions using fitness functions [2]. SBSE can be applied to structural testing, model-based testing, mutation testing, temporal testing, exception testing, configuration and interaction testing, stress testing, regression testing, and integration testing [2]. Regression testing is used to test a modified version of a program [3]. It is expensive because engineers may start testing the modifed version by retesting the test suite that is used to test the old program version.

Minimization techniques aim to reduce the size of the regression test suite of a software system untill it reaches a state where it may be no longer feasible to execute the entire test suite. Prioritization techniques generate an ideal test execution order. Regression

testing requires optimization to solve the case when there are large data sets [3]. The test suite minimization problem is crucial due to test time and resource constraints [4]. It aims to achieve the smallest subset of test cases that covers testing all the requirements and will be discussed in detail later in the paper [4, 3]. Researchers have discussed the classic test suite minimization problem in different ways. These ways can be classified as evolutionary algorithms and nonevolutionary algorithms. The nonevolutionary algorithms started in 1998 when Chen and Lau proposed this problem using the greedy algorithm, but it involved the random selection of test cases in small scale test suites and required optimization in large scale cases [5]. In 2004, J. Black, E. Melachrinoudis and D. Kaeli proposed a bicriteria model for minimizing test cases to reveal the errors, but it only minimizes the test suite with respect to their fault detection effect [6]. Sriraman and Neelam also proposed a concept analysis inspired greedy algorithm for test suite minimization in 2005, but their minimized results are not always smaller than the input suites [7]. In 2006, Khan and Nadeem used statement-coverage criterion to propose the Test Filter with the drawback of wasting time and costs when applying the coverage [8]. In 2007, Jeffrey and Gupta uses selective redundancy to generate a representative set for the test suite reduction, but they need to select tests that expose additional faults in the software [9]. In 2009, Hwa-You and Alessandro used MINTS to propose a general test suite minimization framework and tool, but they introduced fairly similar results for different versions of a program, which requires more investigation [10]. Saeed Parsa and Alireza Khalilian also used the greedy algorithm in their optimization approach of test suite minimization in 2010, but their algorithm requires the coverage information for a single criterion, and it needs to be tested with more than one criterion [11]. In 2010, Yoo and Harman proposed multiobjective test suite reduction, but it needs more optimization for better results and they suggest combining various techniques together, such as the efficient approximation of the greedy approach with a population-based genetic algorithm [12].In 2012, Khalilian and Parsa proposed bicriteria test reduction with cluster analysis of execution profiles, but it used two coverage criteria to improve the fault detection's effectiveness [13]. In 2013, Ankur Prakash proposed a model based on the Boolean function simplification, but it is tested over a maximum of 9 test cases in a test suite, which is quite small compared to benchmark applications [14]. In 2014, Isha used the concepts of set theory to propose their minimization technique, but it was not implemented [15]. In 2017, an integer linear programming model was proposed for this issue, but it was applied to small scale tests [16]. Additionally, Shilpi and Raj proposed a multi criteria based test suite optimization framework; however, it just minimizes the test suite, but it does not produce the optimal solution [17].
In [6] researchers used the Linear Formulation with the Linear Solver (LF_LS) approach to solve the Multi-Criteria Test Suite Minimization (MCTSM) problem by modeling the problem using a linear formulation and solving it with a linear solver. MCTSM can also use a nonlinear solver, which is then referred to as the Nonlinear Formulation with Nonlinear Solver (NF_NS) but it does not guarantee optimal solutions. Auxiliary variables can be used to change the nonlinear MCTSM to a linear one, which can be referred to as the nonlinear formulation using linear solvers (NF_LS) [18]. The experimental results of the proposed technique will be compared to those techniques, as will be shown later in Section 4.

Genetic algorithms were first proposed by John Holland in the 1970s. They have various applications such as software testing, computer-automated design, code-breaking, optimization, image processing, quality control, feature selection for machine learning, filtering and signal processing, finding hardware bugs, the travelling salesman problem and its applications, and many others [4]. Solving the test suit minimization problem using evolutionary algorithms started in 2001 when Lou and Lu proposed a genetic algorithm for the time-aware regression testing reduction problem that examines some criteria to confirm their satisfaction, but their experimental results show that sometimes a vector-based reduction strategy performs better than their proposed genetic one [1]. In 2005, a genetic algorithm was proposed that builds the initial population based on test history, but the problem is that it needs its fault detection capability and other criteria examined [19]. In 2015, Sudhir Kumar proposed an ant colony optimization algorithm for the test case reduction of object oriented programs, but the resulting reduction rates were improved in other research papers [20]. While a genetic algorithm with varying chromosome lengths was implemented to solve the minimization problem by Sudhir and Srinivas in 2015, their algorithm takes quite a lot of time compared to other genetic algorithms [21]. Another algorithm that was implemented in 2017 combines the greedy and genetic algorithms, which gives better results than the greedy algorithm, but the execution time of the GA exceeds the greedy one [22].
There are other evolutionary algorithms such as Particle Swarm Optimization (PSO) that describes the behavior of separation, alignment, and cohesion. Here, separation means to avoid the crowded local flock mates, alignment means moving towards its average

direction, and cohesion means moving towards its average position. However, when the PSO is applied to the test suite minimization problem and finds an optimal point, then other particles will be close to that point, resulting in a weak global search and the inability to find the minimum number of test cases [23]. Ant Colony Optimization (ACO) can be used to solve optimization problem, but it does not always achieve good results. ACO is inspired by the system by which ants follow a marked path with high intensity, which is not the case in our problem because we have equal priorities for all the requirements [23]. Differential Evolution (DE) and the GA are similar except for that DE depends more on mutation, unlike the GA, which depends more on crossover operations [23]. DE can be used in global optimization problems and quantum computing principles can be used to enhance its performance [24]. Genetic algorithms can be implemented on quantum computers since the universal Quantum Turing Machine (QTM) was proposed in 1985 [25]. A Universal Quantum Simulator has been proved to be possible, which makes anything that is computable using a classical computer to also be computable using a quantum computer [26]. Deutsch also introduced quantum parallelism, which can be considered as a key of most successful algorithms. In 1994, Shor introduced an algorithm that is composed of a QTM and a TM to solve the factorization problem in polynomial time [26]. The QIGA uses a hybrid strategy to incorporate quantum computation concepts into the classical genetic algorithm (GA), which combines some common operations such as crossover and mutation in the classical GA with quantum characteristics such as a quantum rotation gate. This combination improves the performance of some classical techniques, as shown in Fig. 2 in [27]. There are many quantum-inspired genetic algorithms in the literature e.g., [23]. Quantum-inspired Evolutionary Algorithms (QEAs) have been successfully applied to solve knapsack problem, the travelling salesman problem, the N-Queens problem, the job shop scheduling problem [28], and others. It has been applied also to solve optimization problems in networking and communication. QEAs also explore the search space very well due to the diverse results from the states' superposition.

The aim of this paper is to solve the test suite minimization problem using a quantum-inspired genetic algorithm where the suggested number of tests based on the chromosome length can be reduced by quantum measurements. The proposed algorithm takes a novel approach regarding the main components of the evolutionary algorithm where local and global parameters are considered simultaneously. In addition to the proposed fitness function that considers the local and the global parameters, better results are obtained using the proposed crossover that affects the local parameters without affecting the global parameters. Using quantum interference and quantum measurement in the proposed algorithm leads to faster convergence.

The remainder of this paper is organized as follows. Section 2 defines the test suite minimization problem, reviews the background concepts of the QIGA, and gives a review of the basic concepts of quantum computing and the operations of the QIGA. In Section 3, the proposed technique is illustrated by defining the encoding, the fitness function, the selection operator, the crossover operator, the mutation operator, and the interference operator. In Section 4, the proposed algorithm is evaluated and the experimental results are given. Finally, Section 5 concludes the paper.

Table 1: Example for test suite minimization problem

TestNo	R_1	R_2	R_3	R_4	R_5	R_6
T_1	0	1	0	0	1	1
T_2	1	0	1	0	0	0
T_3	1	0	0	0	1	0
T_4	1	0	1	1	0	0

2 Background

2.1 Test Suite Minimization Problem

The test suite minimization problem is a crucial problem since it affects time and resources constraints. The problem can be defined as follows.
Take a test suite T with a set of n test cases $\{t_1, t_2, t_3, ..., t_n\}$ and a set R of m test requirements $\{R_1, R_2, R_3, ..., R_m\}$. Each test case t_i, where $1 \leq i \leq n$ covers a subset S_i of the test requirements, such that $S_i \subseteq R$, and $0 < |S_i| \leq m$. It is required to find the minimal subset of T that covers all the test requirements [4]. For example, Table 1 illustrates a given test suite showing the requirements that are covered with each test case. Many solutions can be found that cover all the requirements, but the target here is to find the minimum number of tests for these solutions. For example, all the requirements can be covered with the test set $\{T_1, T_2, T_4\}$ or $\{T_1, T_3, T_4\}$, but the minimum set is $\{T_1, T_4\}$. It becomes more complicated with large data sets. The TestNo column represents the test case number while the Rs columns represent the requirements that are to be satisfied by each test case.

To solve this instance of the test suit minimization problem that is shown in Table 1, the minimal

number of true assignments that satisfy the following Boolean formula should be found as follows:

$$f_{TR} = \underbrace{(T_2 \vee T_3 \vee T_4)}_{R_1} \wedge \underbrace{(T_1)}_{R_2} \wedge \underbrace{(T_2 \vee T_4)}_{R_3} \wedge \underbrace{(T_4)}_{R_4} \wedge \underbrace{(T_1 \vee T_3)}_{R_5} \wedge \underbrace{(T_1)}_{R_6}, \quad (1)$$

which is a reduction of the test suite minimization problem to the SAT problem.

2.2 Quantum-Inspired Genetic Algorithm (QIGA)

2.2.1 Quantum Computing

The unit of information in a quantum computer is the quantum bit or qubit, which can be in one of two states, $|0\rangle$, or $|1\rangle$ as well as a linear combination of both states (superposition principle). This linear combination represents a qubit in its quantum superposition as follows [29]:

$$|\psi_1\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad (2)$$

where $\alpha and \beta$ are complex numbers such that

$$|0\rangle = [1 \quad 0]^T\,, |1\rangle = [0 \quad 1]^T\,, \quad (3)$$

and

$$|\,\alpha\,|^2 + |\,\beta\,|^2 = 1. \quad (4)$$

In general, the quantum computation can be represented as follows:

$$F\,|X\rangle = |\psi\rangle\,, \quad (5)$$

where $|X\rangle$ represents the initial system state and F is a unitary operator that is applied to the state $|X\rangle$ resulting in $|\psi\rangle$ which is the final state that is achieved [30].

Quantum computers store the data in registers as quantum states that are the tensor product of two or more qubits [28]; for example, the tensor product of two qubits is calculated as follows:

$$\begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix} \otimes \begin{bmatrix}\alpha_2\\ \beta_2\end{bmatrix} = \begin{bmatrix}\alpha_1\alpha_2\\ \alpha_1\beta_2\\ \beta_1\alpha_2\\ \beta_1\beta_2\end{bmatrix}. \quad (6)$$

The special states that cannot be written as a tensor product of its components are known as the entangled states [30].

Quantum gates perform unitary transformations and are represented by matrices. One important feature of quantum gates is their reversibility. An n-input Boolean function is called reversible if the following hold: [31]

1. The number of outputs and inputs are the same, and
2. Any input pattern maps to a unique output pattern.

One of the quantum gates is the Hadamard or H gate [28],

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}, \quad (7)$$

which has the following effect on a qubit:

$$H.\,|\psi_1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \cdot \begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}\alpha+\beta\\ \alpha-\beta\end{bmatrix}. \quad (8)$$

For example, we have the following:

$$H.\,|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\ 0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix}. \quad (9)$$

$$H.\,|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix}. \quad (10)$$

One of Hadmard's gate applications is to initialize a quantum register because applying $H^{\otimes n}$ gates on a quantum register of n qubits that are initialized to state $|0\rangle$ gives a superposition of all the 2^n possible states as follows:

$$H^{\otimes n}\,|0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle\,. \quad (11)$$

There are many quantum gates such as CNOT, Toffoli, and Fredkin gates as follows:

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}. \quad (12)$$

$$Toffoli = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \quad (13)$$

$$Fredkin = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (14)$$

The quantum rotation gate, $U(\theta)$, helps to update the quantum state, and its function is as follows:

$$U(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad (15)$$

where θ is the rotation angle.

2.2.2 Operations of the QIGA

The quantum-inspired genetic algorithm uses a representation that is based on the concept of qubits. One qubit is defined using a pair of complex numbers, (α, β), as follows [32]:

$$|\psi_1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}. \quad (16)$$

Considering that an m-qubits representation is defined as follows:

$$|\psi_m\rangle = \begin{bmatrix} \alpha_1|\alpha_2|\alpha_3|.....|\alpha_m \\ \beta_1|\beta_2|\beta_3|.....|\beta_m \end{bmatrix}, \quad (17)$$

where $|\ \alpha_i\ |^2 + |\ \beta_i\ |^2 = 1, i = 1, 2, 3, ..., m$. This representation can represent a superposition of states. For instance, assume that there is a three-qubits system with three pairs of amplitudes such as follows:

$$|\psi_3\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}}|\frac{1}{\sqrt{2}}|\frac{1}{2} \\ \frac{1}{\sqrt{2}}|\frac{-1}{\sqrt{2}}|\frac{\sqrt{3}}{2} \end{bmatrix}. \quad (18)$$

Then, the system state can be represented as follows [28]:

$$|\psi_3\rangle = \frac{1}{4}|000\rangle + \frac{\sqrt{3}}{4}|001\rangle - \frac{1}{4}|010\rangle - \frac{\sqrt{3}}{4}|011\rangle + \frac{1}{4}|100\rangle + \frac{\sqrt{3}}{4}|101\rangle - \frac{1}{4}|110\rangle - \frac{\sqrt{3}}{4}|111\rangle. \quad (19)$$

The advantage of the quantum-inspired genetic algorithm is that a quantum population can be exponentially larger than a classical one of the same size because a quantum chromosome can exploit the superposition to represent an exponentially large number of classical chromosomes simultaneously.

- **Rotation (Interference) Gate**

The rotation operator or quantum interference is a gate $U(\theta)$ similar to that shown in (15), which has the following effect when applied on a qubit:

$$U(\theta)|\psi_t\rangle = |\psi_{t+1}\rangle = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix} = \begin{bmatrix} cos(\theta)\alpha_t - sin(\theta)\beta_t \\ sin(\theta)\alpha_t + cos(\theta)\beta_t \end{bmatrix}. \quad (20)$$

- **Quantum Mutation**

The mutation applies a random change to a chromosome with a certain mutation rate. Mutation is applied by randomly selecting a mutation point in the chromosome and replacing that point with another random value from a given set of values. For example, assuming the following chromosome is given and the mutation is applied on the first qubit, then a randomly chosen value is used to replace that first qubit but without violating the quantum state rule that is shown in (4), as follows:

$$p = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & ... & \alpha_q \\ \beta_1 & \beta_2 & \beta_3 & ... & \beta_q \end{bmatrix}. \quad (21)$$

The new chromosome will be as follows:

$$p' = \begin{bmatrix} \alpha'_1 & \alpha_2 & \alpha_3 & ... & \alpha_q \\ \beta'_1 & \beta_2 & \beta_3 & ... & \beta_q \end{bmatrix}, \quad (22)$$

where

$$| \alpha_1' |^2 + | \beta_1' |^2 = 1. \quad (23)$$

- **Quantum Crossover**

The quantum-inspired version of the classical crossover operator is applied in many practical optimization problems. For the one point crossover, for example, if the cut point is randomly chosen to be a point between the first and second positions, then an exchange of chromosomal segments between chromosomes p_1 and p_2 is as follows:

$$p_1 = \begin{bmatrix} \alpha_1 | & \alpha_2 & \alpha_3 & ... & \alpha_q \\ \beta_1 | & \beta_2 & \beta_3 & ... & \beta_q \end{bmatrix}, \quad (24)$$

$$p_2 = \begin{bmatrix} \alpha_1' | & \alpha_2' & \alpha_3' & ... & \alpha_q' \\ \beta_1' | & \beta_2' & \beta_3' & ... & \beta_q' \end{bmatrix}. \quad (25)$$

This will result in the following:

$$p_1' = \begin{bmatrix} \alpha_1 & \alpha_2' & \alpha_3' & ... & \alpha_q' \\ \beta_1 & \beta_2' & \beta_3' & ... & \beta_q' \end{bmatrix}, \quad (26)$$

and

$$p_2' = \begin{bmatrix} \alpha_1' & \alpha_2 & \alpha_3 & ... & \alpha_q \\ \beta_1' & \beta_2 & \beta_3 & ... & \beta_q \end{bmatrix}. \quad (27)$$

3 The Proposed Technique

This section illustrates the detailed structure and steps to solve the test suite minimization problem using the QIGA.

3.1 Problem Representation

To solve the test suite minimization problem, a test requirement matrix (TR) should be given to show what requirements are covered in each test. In this paper, various matrices are tested in addition to the identity matrix, as will be seen in the results section. It can be any binary matrix. Each row represents a test while each column tells whether or not a specific requirement is considered in this test. It is a binary matrix, the value of 1 means that the requirement is fulfilled in the test while the value of 0 means that the requirement is not covered. Table 1 shows a test case example with the requirements that each test case satisfies. The aim is to find the minimum number of test cases that satisfies all the requirements. The matrix that is derived from Table 1 is as follows:

$$TR = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}. \quad (28)$$

Table 2: Example to illustrate the global α and global β values

TestNo	Global Values
T_1	α_1, β_1
T_2	α_2, β_2
T_3	α_3, β_3
T_4	α_4, β_4

An important point to consider here is the chromosome data structure. The chromosome is composed of a set of tests that are chosen from the tests that are given in the TR. For example, if the chromosome size is 3, this can be (T_1, T_3, T_4). There are globally calculated α and β values that are associated with each test in the derived matrix. These global α and global β values are fixed for this given matrix (TR). Table 2 shows an example for the various tests existing in the given matrix along with the α and β values that are globally associated with them. Each population includes a number of tests that are selected from the TR, and each of these tests has its calculated global α and β. When they are selected for the chromosome, the local α and β values will be initialized using the global α and β. These local α and β are updated from one generation to another based on applying the interference operator on the chromosome, as calculated in (20). Table 3 shows how the local α and β are attached to the tests. For example, in chromosome 1, the chromosome contains $T1$, $T2$, and $T4$. The amplitudes of T_1 have been initialized with the global α and β and the amplitudes have been updated using the rotation operator to reach α_{p11} and β_{p11} (the local α and β). The indices of the local α and β are composed of three parts, including the population number, the chromosome number, and the test, respectively.

The population size that is used in the QIGA to solve the given test is $popSize$. $sumOfOnes$ and $sumOfAllOnes$ representing the summation of 1's with their weights and the summation of the case when there is a string of all 1's, respectively. These values need to be prepared as follows:

$$sumOfOnes = \sum_{j=0}^{testColSize} ((testColSize - (j)) * (j+1) * elementValue, \quad (29)$$

$$sumOfAllOnes = \sum_{j=0}^{testColSize} (j+1) * (testColSize - j), \quad (30)$$

Table 3: Example showing the local α and β values in a specific population p of size 4

Chromosome Number	Chromosomes(population, chromosome, test)		
1	T1 $\alpha_{p11}, \beta_{p11}$	T2 $\alpha_{p12}, \beta_{p12}$	T4 $\alpha_{p14}, \beta_{p14}$
2	T3 $\alpha_{p23}, \beta_{p23}$	T1 $\alpha_{p21}, \beta_{p21}$	T2 $\alpha_{p22}, \beta_{p22}$
3	T3 $\alpha_{p33}, \beta_{p33}$	T1 $\alpha_{p31}, \beta_{p31}$	T4 $\alpha_{p34}, \beta_{p34}$
4	T1 $\alpha_{p41}, \beta_{p41}$	T2 $\alpha_{p42}, \beta_{p42}$	T3 $\alpha_{p43}, \beta_{p43}$

where $testColSize$ is the column size of the test requirement matrix, and the $elementValue$ is the value of each added element such that $elementValue \in \{0, 1\}$, which means that the $elementValue = 0$ can be ignored. The global α and β can be initially calculated as follows:

$$global\_\beta = \sqrt{sumOfOnes/sumOfAllOnes}, \quad (31)$$

$$global\_\alpha = \sqrt{1.0 - global\_\beta^2}. \quad (32)$$

The initial population is then generated. A number of chromosomes are then randomly generated using the tests in the test requirement matrix. Interference is then applied to update the local α and β values for each chromosome i, as in (20), where the change in θ will be a random number between 0 and 1, which is represented by $\delta\theta$, as shown in (33). Then, the data are measured based on the updated local α and β values.

$$\theta_{t+1} = \theta_t \pm \delta\theta_t. \quad (33)$$

3.2 Fitness Function

To calculate the fitness function, a weight is prepared for the whole matrix using the $sumOfAllOnes$ and is calculated as in (30). Then, a weight for each chromosome in the population is prepared and denoted as w, as follows:

$$w = \sum_{j=0}^{testColSize} ((testColSize-(j))*(j+1)). \quad (34)$$

Then, the w values are used to calculate the fitness function as follows:

$$fitnessValue = \frac{|w|}{sumOfAllOnes} * 100. \quad (35)$$

3.3 Selection

The fitness function is calculated for each chromosome and the Roulette wheel algorithm is used for selecting chromosomes from the random initial population.

3.4 Crossover

A single point crossover is then applied with a crossover probability of 90%. The crossover operation considers the local α and β values but it does not affect the global α and β that are associated with the TR tests. The operation first passes by each chromosome with a random cross rate between 0 and 1 to mark the chromosomes that will apply crossover. A random crossover position is then selected and the operation is applied.

3.5 Mutation

The mutation is applied at a rate of 5% by randomly choosing a point in a chromosome and replacing it using a random test from the TR.

3.6 Interference

The interference is applied to all tests in the chromosome. It updates the θ value with a random number $\delta\theta$ between 0 and 1, as shown in (33). Then, it updates the local α and β values according to (20).

4 Experimental Results

The pseudo code that is shown in Algorithm 1 summarizes the proposed method. To evaluate the proposed method, experiments has been conducted on random test suite minimization problems with different sizes. Table 4 summarizes the experiments' GA parameters. For demonstration purposes, the results of the case with 200 test cases and 200 test requirements, i.e., a matrix of size 200 $\times$ 200, will be discussed. Three different types of matrices have been randomly generated: the sparse matrices where the 1s randomly occupy 20% of the matrices, the balanced matrices where the 1s randomly occupy 50% of the matrices, and the dense

Algorithm 1 Pseudocode for the proposed algorithm

Read a test requirement matrix TR of size $m \times n$.
Calculate $sumOfOnes$ and $sumOfAllOnes$ values for the whole matrix as illustrated in (29) and (30).
Generate the initial population of a chosen population size and a chosen chromosome length. The elements of the chromosomes in each population are chosen from the tests in the test requirement matrix.
Initialize the local α and β values from the global ones.
while a 100% fitness value is reached or a predefined number of iterations is reached **do**
Apply the interference operation and then the measurement.
Update the local α and β values according to the interference operation.
Select from the population using Roulette wheel method.
Apply crossover with a crossover rate of 90%.
Apply mutation with a mutation rate 5%.
Calculate the fitness values for the population elements using the fitness function in (35).
Check the maximum fitness among the population elements.
if a 100% fitness value is found **then**
Report a solution that covers all the requirements that are found.
end if
end while

matrices where the 1s occupy 80% of the matrices.

The same random test suite minimization problem has been solved using the classical GA and the proposed QIGA with similar parameters such as the mutation rate, the population size, and the crossover rate to be able to compare their performances. The chromosome length has been fixed for 100 generations or until a fitness of 100% is reached. If no solution was found using the assumed chromosome length, then the size of the chromosome will be gradually incremented and then fixed for another 100 generations and so on, until the maximum length of the chromosome, which is equal to the number of tests, is reached. This will help to demonstrate the ability of the proposed QIGA to discover the minimum number of test suites despite the suggested chromosome length, as in the case of the classical GA, and this will be shown next.

Using the same chromosome length, both the classical GA and the proposed QIGA evolved to get a solution. The found solution using the classical GA can be further reduced by eliminating repeated test cases (redundancy). In the case of the QIGA, the reduced number of the test cases is obtained by the quantum measurement where only a subset of the test cases is assumed to be the candidate solutions based randomly on the amplitudes of the superposition. Fig. 1 shows a comparison between the size of the obtained solution using the classical GA and the proposed QIGA. This shows that even by increasing the suggested size of the chromosome, the proposed QIGA can achieve a better reduction in the size of the solution compared with the classical GA. This can help to remove the burden of assuming a size for the chromosome. A large chromosome size can be used and the quantum measurement can help to reduce the solution size to the near minimal size.

The proposed algorithm used the minimum chromosome length, i.e., the minimum number of test cases. The average fitness and the maximum fitness of the population have been recorded during the evolution for the three cases, i.e., sparse, balanced and dense. Then, the average performance over 50 trials is used in the analysis, as shown in Fig. 2, Fig. 3 and Fig. 4, respectively, which show that the proposed QIGA achieves faster convergence than the classical GA. Fig. 2 analyzes the sparse case by calculating the average fitness values over 50 runs, as shown by the QIGA_avg curve, and similarly the GA_avg is calculated. This comparison shows faster convergence for the QIGA_avg curve; moreover, it reaches higher fitness values than the GA_avg, even before the convergence. To give more power and consistency to the results, the maximum fitness values are also considered, as shown in the QIGA_max. Then, it is compared to the maximum fitness values of the GA_max curve. This comparison confirms the faster convergence of the QIGA algorithm than the GA algorithm and it shows also higher fitness values than the GA ones before the convergence. The other case to be studied is the balanced case, which is analyzed in Fig. 3, where QIGA_max represents the maximum fitness values that are generated by applying the QIGA algorithm. The QIGA_max clearly converges faster than GA_max, which shows the maximum fitness values that are generated by applying the GA algorithm. Then, the average fitness values are considered for both the QIGA and GA algorithms, as shown in the QIGA_avg and GA_avg curves. QIGA_avg gives better results than GA_avg. Fig. 4 shows the maximum fitness values that are calculated in the dense case using the QIGA_max curve and compares them to the ones that are calculated in the dense case using the GA algorithm with the GA_max curve. Here, QIGA_max converges faster and has better fitness values than GA_max. Similarly, the average fitness values are studied and shown using QIGA_avg and

GA_avg, which proves the better convergence of the QIGA algorithm than the GA algorithm.

Table 5 provides the summary of the data set that is used in the experiment [18, 33]. It shows the information such as the version, description, and LOC, which represents the lines of code in the program. Then, it also provides columns for the number of test cases before the reduction, the #Tests for the reduced suite size, and the #Faults for the number of detected faults in the program, which measures the fault detection ability. Table 6 and Table 7 compare the experimental results using the proposed technique over the five programs in the data set with the results that are shown in [18], which shows a better reduction in the number of required tests. Table 6 displays the test suite size before and after the reduction for the Grep, Flex, and Sed programs. The results of applying the LF_LS technique, NF_LS technique, NF_NS technique, and the proposed technique are shown for each program and then a comparison is performed between the proposed technique and each of the mentioned techniques. This illustrates the better results of the proposed technique than the others. Similarly, Table 7 gives the same study for the Make and Gzip programs and it also illustrates the better reduction for the proposed technique than NF_LS, LF_LS, and NF_NS.

5 Conclusion

In this paper, a quantum-inspired genetic algorithm (QIGA) has been proposed to solve the test suite minimization problem where quantum superposition has been used in the encoding of the chromosome to increase the size of the search space over approximately the same physical space. The quantum rotation gate has been used with the crossover and mutation operators to enhance the search capabilities of the classical genetic algorithm. The proposed algorithm used local and global parameters simultaneously to guide the search. Here, the crossover operator affects the local parameters, while the global parameters have been kept unchanged to save the algorithm from getting lost due to the increase in the search space because of the quantum superposition. The quantum measurement has been be used in the proposed quantum-inspired evolutionary algorithm to reduce the number of test cases to avoid any priori assumptions on the minimal number of test cases. It has been shown that the adopted quantum techniques accelerates the convergence to the solution compared with the classical genetic algorithm.

Experimental results have been shown for the sparse, balanced and dense instances of the test suite minimization problem and the quantum-inspired version performs better than the classical genetic version of the algorithm. The proposed algorithm has been used to solve the instances in the data set that was used in [18, 33]; here, the proposed algorithm provides better reduction results than those of the approaches that are shown in the literature.

Conflicts of interest

conflicts of interest: none

References:

[1] L. You, Y. Lu, A genetic algorithm for the time-aware regression testing reduction problem, *The 2012 eighth International Conference on Natural Computation*.

[2] A. Orso, G. Rothermel, Software testing: a research travelogue (2000:2014), Proceedings of the on Future of Software Engineering - FOSE14 (2014) 117–132.

[3] S. Yoo, M. Harman, Regression testing minimisation, selection and prioritisation: a survey, Software Testing, Verification and Reliability 22 (2012) 67–120.

[4] M. Harman, P. McMinn, J. T. de Souza, S. Yoo, Search based software engineering: techniques, taxonomy, tutorial, LASER Summer School on Software Engineering (2012) 1–59.

[5] T. Y. Chen, M. F. Lau, A new heuristic for test suite reduction, Information and Software Technology 40 (1998) 347–354.

[6] J. Black, E. Melachrinoudis, D. Kaeli, Bi-criteria models for alluses test suite reduction, International Conference on Software Engineering.

[7] S. Tallam, N. Gupta, A concept analysis inspired greedy algorithm for test suite minimization, PASTE '05 Proceedings of the 6th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering (2005) 35–42.

[8] S. Khan, A. Nadeem, Testfilter: A statement-coverage based test case reduction technique, The 2006 IEEE International Multitopic Conferencedoi:10.1109/inmic.2006.358177.

[9] D. Jeffrey, N. Gupta, Improving fault detection capability by selectively retaining test cases during test suite reduction, IEEE Transactions on Software Engineering 33 (2007) 108–123.

[10] H. Hsu, A. Orso, Mints: A general framework and tool for supporting test-suite minimization, International Conference on Software Engineering.

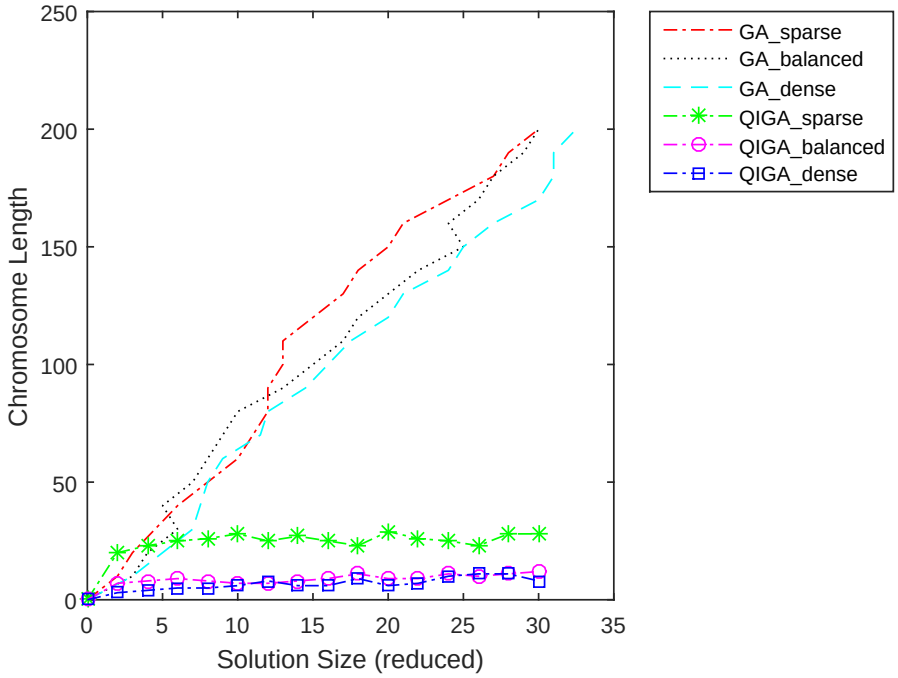

Fig. 1: Reduction in the size of the solution using the classical GA and the proposed QIGA. Here, the reduction in the classical GA is obtained by eliminating the redundancy while the reduction in the QIGA is obtained by the quantum measurement.

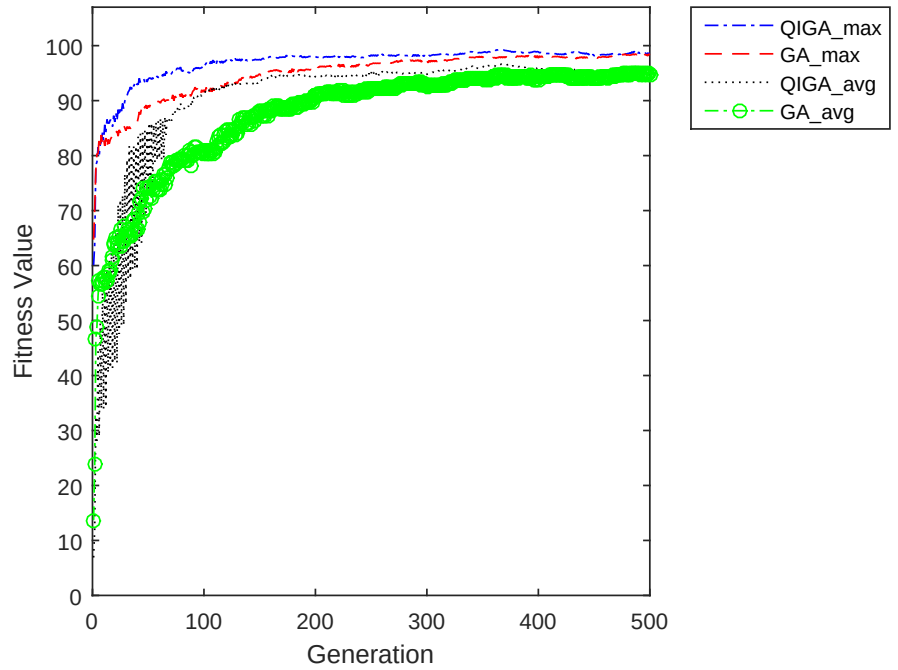

Fig. 2: The convergence of the average fitness values and the maximum fitness value of the proposed QIGA compared with the classical GA for the sparse case.

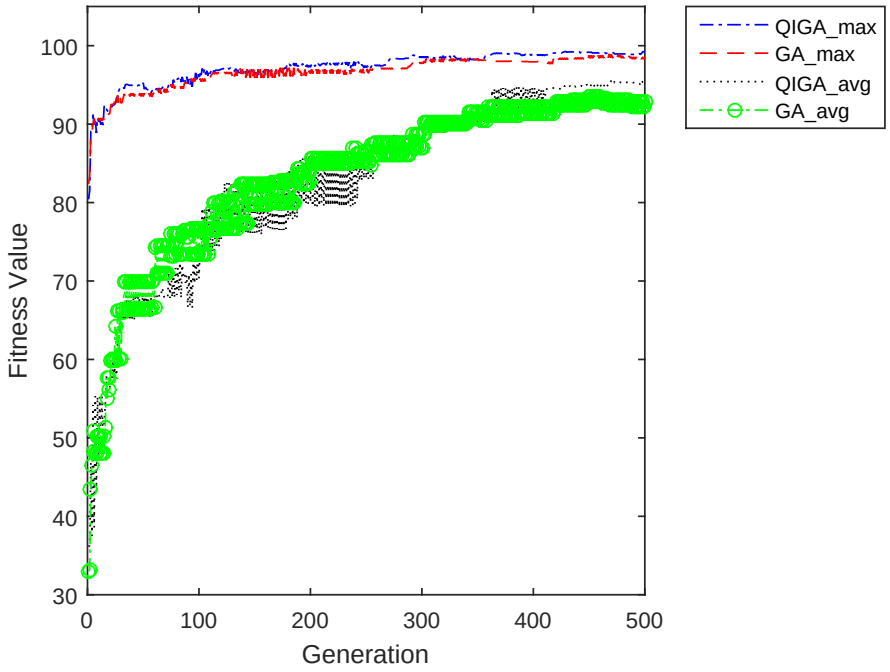

Fig. 3: The convergence of the average fitness values and the maximum fitness value of the proposed QIGA compared with the classical GA for the balanced case.

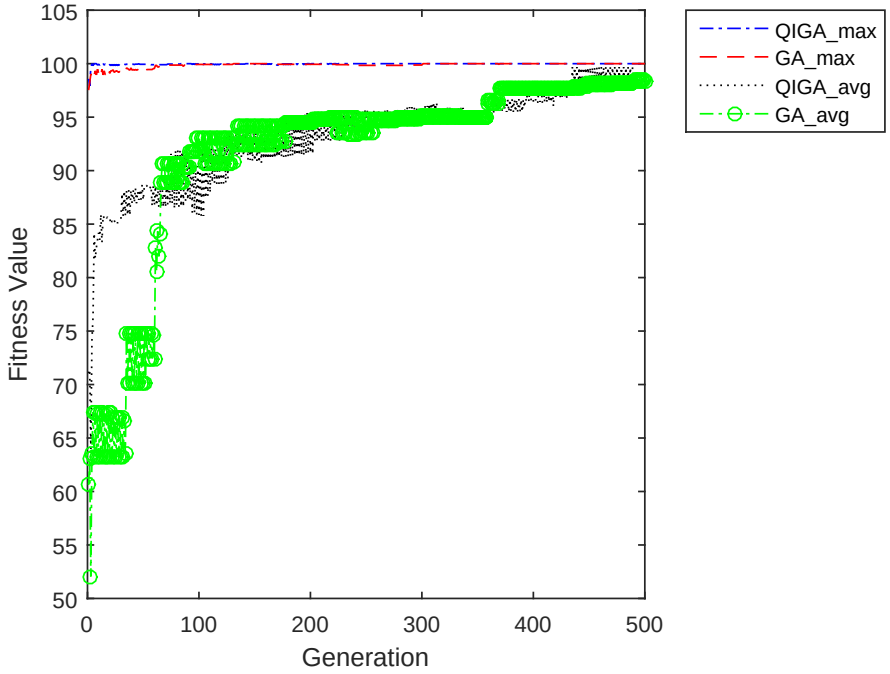

Fig. 4: The convergence of the average fitness values and the maximum fitness value of the proposed QIGA compared with the classical GA for the dense case.

Table 4: GA Parameters for the proposed technique

GA Parameter	Value
Population Size	500
Crossover	Single-point
Crossover Rate	90%
Mutation Rate	5%
Selection	Roulette Wheel
$\theta_{initial}$	π
$\delta\theta$	A random number between 0 and 1
Operator Probabilities	Selected with trials
Termination	To find a solution OR Reach the number of trials

Table 5: Programs used in the experimental results [18].

Program	Version	Description	LOC	#Tests	#Faults
Grep	2.7	Pattern search and matching utility	58,344	746	54
Flex	2.5.4	Lexical analyzer	12,366	605	37
Sed	4.2	Command line text editor	26,466	324	25
Make	3.80	Executable builder and generator	23,400	158	15
Gzip	1.3	Data compressor	5,682	397	56

Table 6: Comparison among the different methods modeling the classic bicriteria problem.

Programs	Grep		Flex		Sed	
	#T	#F	#T	#F	#T	#F
Methods						
Original	746	54	605	37	324	25
LF_LS [18]	59	29	44	28	12	21
NF_LS [18]	59	36	44	32	12	25
NF_NS [18]	n/a	n/a	44	32	12	25
Proposed Technique	42	30	32	21	10	14
%Proposed_technique over NF_LS	40.47%	20%	37.5%	52.38%	20%	78.57%
%Proposed_technique over LF_LS	40.47%	3.45%	37.5%	33.33%	20%	50%
%Proposed_technique over NF_NS	n/a	n/a	37.5%	52.38%	20%	78.57%

Table 7: Continuation of Table 6: Comparison among the different methods modeling the classic bicriteria problem.

Programs	Make		Gzip	
	#T	#F	#T	#F
Methods				
Original	158	15	397	56
LF_LS [18]	14	12	45	50
NF_LS [18]	14	13	45	50
NF_NS [18]	14	13	45	49
Proposed Technique	10	10	32	44
%Proposed_technique over NF_LS	40%	30%	40.6%	12.24%
%Proposed_technique over LF_LS	40%	20%	40.6%	12.24%
%Proposed_technique over NF_NS	40%	30%	40.6%	2.04%

[11] S. Parsa, A. Khalilian, On the optimization approach towards test suite minimization, International Journal of Software Engineering and Its Applications 4.

[12] S. Yoo, M. Harman, Using hybrid algorithm for pareto efficient multi-objective test suite minimization, The Journal of Systems and Software 83 (2010) 689–701.

[13] A. Khalilian, S. Parsa, Bi-criteria test suite reduction by cluster analysis of execution profiles, IFIP International Federation for Information Processing 2012 (2012) 243–256.

[14] R. Singh, M. Santosh, Test case minimization techniques: A review, International Journal of Engineering Research & Technology 2.

[15] I. Mangal, D. Bajaj, P. Gupta, Regression test suite minimization using set theory, International Journal of Advanced Research in Computer Science and Software Engineering 4.

[16] S. Panda, D. P. Mohapatra, Regression test suite minimization using integer linear programming model, Software: Practice and Experience 47 (2017) 1539–1560.

[17] S. Singh, R. Shree, A multi criteria based test suite optimization framework, International Journal of Software Engineering and Its Applications 11 (2017) 77–86.

[18] J. Lin, J. Garcia, R. Jabbarvand, S. Malek, Nemo: multi-criteria test-suite minimization with integer nonlinear programming, The Proceedings of ICSE '18: 40th International Conference on Software Engineering (2018) 1039–1049.

[19] X. Ma, B. Sheng, C. Ye, Test-suite reduction using genetic algorithm, The Proceedings of the sixth international conference on Advanced Parallel Processing Technologies (2005) 253–262.

[20] S. Mohapatra, M. Pradhan, Finding representative test case for test case reduction in regression testing, Intelligent Systems and Applications 7 (11) (2015) 60–65.

[21] M. Alian, D. Suleiman, A. Shaeout, Test case reduction techniques - survey, International Journal of Advanced Computer Science and Applications 7 (2016) 264–275.

[22] A. Yamuc, M. O. Cingiz, G. Biricik, O. Kalipsiz, Solving test suite reduction problem using greedy and genetic algorithms, International Conference on Electronics, Computers and Artificial Intelligence (ECAI).

[23] A. M. Adrian, A. Utamima, K. Wang, A comparative study of GA, PSO and ACO for solving construction site layout optimization, KSCE Journal of Civil Engineering 19 (2014) 520–527.

[24] Y. Wang, Y. Li, L. Jiao, Quantum-inspired multi-objective optimization evolutionary algorithm based on decomposition, Soft Computing - A Fusion of Foundations, Methodologies, and Applications 20 (2016) 3257–3272.

[25] D. Deutsch, Quantum theory, the church-turing principle and the universal quantum computer, Proceedings of the Royal Society A:Mathematical, Physical and Engineering Sciences 400 (1985) 97–117. doi:10.1098/rspa.1985.0070.

[26] C. Zalka, Efficient simulation of quantum systems by quantum computers, Fortschritte der Physik 46 (1998) 877–879.

[27] W. Zhao, S. Guo, Y. Zhou, J. Zhang, A quantum-inspired genetic algorithm-based optimization method for mobile impact test data integration, Computer-Aided Civil and Infrastructure Engineering 33 (2018) 411–422.

[28] R. Lahoz-Beltra, Quantum genetic algorithms for computer scientists, Computers 5 (4).

[29] Y. Zhang, J. Liu, Y. Cui, X. Hei, M. Zhang, An improved genetic algorithm for test suite reduction, IEEE International Conference on Computer Science and Automation Engineering.

[30] N. Toronto, D. Ventura, Learning quantum operators from quantum state pairs, IEEE International Conference on Evolutionary Computation.

[31] S. M. Taha, Reversible logic synthesis methodologies with application to quantum computing, Studies in Systems, Decision and Controldoi:10.1007/978-3-319-23479-3.

[32] H. T. Draa, A new real-coded quantum-inspired evolutionary algorithm for continuous optimization, Applied Soft Computing 61. doi:10.1016/j.asoc.2017.07.046.

[33] C. Henard, M. Papadakis, M. Harman, Y. Jia, Y. L. Traon, Comparing white-box and black-box test prioritization, The IEEE/ACM thirty eight IEEE International Conference on Software Engineering (2016) 523–534.