Reformulating Regression Test Suite Optimization using Quantum Annealing - an Empirical Study

Antonio Trovato[1*], Manuel De Stefano[1], Fabiano Pecorelli[1,2], Dario Di Nucci[1], Andrea De Lucia[1]

[1]SeSa Lab, University of Salerno, Salerno, Italy.
[2]Dep. of Information Science and Technology, Pegaso Digital University, Naples, Italy.

*Corresponding author(s). E-mail(s): atrovato@unisa.it;
Contributing authors: madestefano@unisa.it; fabiano.pecorelli@unipegaso.it; ddinucci@unisa.it; adelucia@unisa.it;

Abstract

Maintaining software quality is crucial in the dynamic landscape of software development. Regression testing ensures that software works as expected after changes are implemented. However, re-executing all test cases for every modification is often impractical and costly, particularly for large systems. Although very effective, traditional *test suite optimization* techniques are often impractical in resource-constrained scenarios, as they are computationally expensive. Hence, *quantum computing* solutions have been developed to improve their efficiency but have shown drawbacks in terms of effectiveness. We propose reformulating the regression test case selection problem to use quantum computation techniques better. Our objectives are (i) to provide more efficient solutions than traditional methods and (ii) to improve the effectiveness of previously proposed quantum-based solutions.
We propose *SelectQA*, a *quantum annealing* approach that can outperform the quantum-based approach *BootQA* in terms of effectiveness while obtaining results comparable to those of the classic *Additional Greedy* and *DIV-GA* approaches. Regarding efficiency, SelectQA outperforms DIV-GA and has similar results with the Additional Greedy algorithm but is exceeded by BootQA.

Keywords: Regression Testing, Quantum Computing, Search-based Software Engineering

1 Introduction

In the ever-evolving landscape of software development, software quality assurance is of fundamental importance. As software systems undergo continuous modifications and enhancements, ensuring these changes do not introduce unintended side effects or defects becomes imperative. In response to this need, regression testing [1] verifies whether previously developed and tested software components still work as expected after a change is performed.

Ideal regression testing would re-run all the available test cases of a given software system. However, in addition to being potentially very costly, this could even be impractical in some cases [2, 3], especially for large systems and when the re-execution of entire test suites occurs for every single modification. Several approaches have been suggested to streamline the regression testing process, such as selecting a potentially minimal subset of test cases from the test suite based on specific testing criteria [4–10]. Alternatively, strategies include prioritizing the execution of test cases, aiming first to run those anticipated to uncover faults earlier in the testing cycle [11–14].

In this scenario, *test case selection* [1, 15] is one of the most widely investigated *test suite optimization* approaches. It consists of choosing a subset of test cases from a pool of possibilities (i.e., the test suite), ensuring comprehensive coverage and efficient use of testing resources.

While the significance of test case selection is undeniable, traditional techniques face serious challenges, primarily related to computing costs. Conventional approaches, often rooted in optimization algorithms, such as greedy algorithms [4] and search-based approaches like DIV-GA [8], demand extensive computational resources to deliver optimal results. The computational burden reduces the efficiency of these methods and poses practical limitations, especially in scenarios where resource-intensive testing is unavailable.

Relying on quantum computing could overcome the limitations of traditional techniques. This new computation technology harnesses the principles of quantum mechanics to process information in ways fundamentally different from classical computers. The intrinsic parallelism and exponential computational capacity of quantum systems offer a potential breakthrough for overcoming the resource constraints associated with traditional test case selection techniques [16, 17].

The current state-of-the-art quantum computing techniques for test suite optimization are represented by *BootQA* [18] and IGDec-QAOA [19]. These techniques represent the first quantum approaches developed for the test suite optimization problem. BootQA resolves the problem through *quantum annealing*, whereas IGDec-QAOA uses the quantum approximate optimization algorithm (QAOA). We focus on quantum annealing as a first step. Developing QAOA strategies is way more challenging due to the significant limitations of current gate-based quantum computers compared to annealing ones. We will perform such a comparison in future work.

This paper proposes SelectQA, a novel quantum computing approach that solves the Test Case Selection (TCS) problem. Like for *BootQA*, the algorithm is implemented through the D-Wave environment [20], which can solve NP-hard combinatorial problems through "*Quantum Simulated Annealing*" [21, 22].

Leveraging Quantum Simulated Annealing [23, 24] and executing different runs of experimentation, SelectQA can produce great (and statistically reliable) results, both in terms of effectiveness and efficiency. In particular, in terms of effectiveness, SelectQA outperforms BootQA while producing results comparable to those of additional greedy and DIV-GA. In terms of efficiency, SelectQA has shown a practically constant total execution time, regardless of the problem size, demonstrating its great scalability. In this perspective, SelectQA outperforms DIV-GA but is outperformed by BootQA. Additional greedy performs faster than SelectQA in two out of four cases, showing its remarkable ability to solve small problems while impractical for larger ones.

These results promise to address the computational challenges plaguing traditional methods. As quantum computing continues to emerge as a driving force in optimization tasks, this work has aimed to provide a significant stride toward advancing the efficiency and effectiveness of regression testing in contemporary software development paradigms, taking additional steps into quantum software engineering research.

Overview of the paper: Section 2 introduces with much more detail the problems of test case selection that SelectQA aims to solve and the methods that represent the current state-of-the-art with which to compare SelectQA; Section 3 describes the actual mathematical formulations behind the work of SelectQA; Section 4 first introduces the goals, metrics, and configurations planned for the experiments; then, it shows the actual results with the correspondent conclusions. Section 5 makes a deeper analysis of all the threats to the validity of this study; finally, Section 6 concludes the paper and envisions future work.

2 Background

This section introduces test case selection, its multi-objective formulation, and the usage of classical and quantum optimizations to solve it.

2.1 Test Case Selection as Optimization Problem

Test case selection focuses on selecting a subset of a test suite to test software changes, i.e., to test whether unmodified parts of a program continue to work correctly after changes involving other parts [25]. Various techniques, such as Integer Programming [26], symbolic execution [27], data flow analysis [28], dependence graph-based methods [29], and flow graph-based approaches [25], can be employed to identify the modified portions of the software. Once test cases covering the unchanged program segments are pinpointed using a specific technique, an optimization strategy can *select a minimal set of these test cases based on certain testing criteria* (e.g., branch coverage). The ultimate aim is to reduce the costs associated with regression testing.

While initially test case selection was considered a single-objective optimization problem, advancements in the method revealed that optimizing only one objective is insufficient, as many tests often need to satisfy multiple criteria simultaneously. Therefore, Yoo et al. [4] introduced the idea of using Pareto sets to address a multi-objective version of the test case selection problem. In particular, in their multi-objective variant, the goal is to build a *Pareto-efficient* subset of the starting test suite, leveraging different selection criteria, given a set of components to test.

Let $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ be the starting test suite and let $F = \{f_1, f_2, ..., f_m\}$ the set of objective functions that mathematically formalize the criteria for the selection process. The multi-objective formulation of the test case selection problem can be seen as selecting a subset $\Gamma' \subseteq \Gamma$ such that Γ' is the Pareto-optimal set concerning the objective functions in F.

What emerges from the definition above is that the optimality of a solution is measured using the concepts of Pareto optimality and dominance. A solution X is said to be *Pareto-optimal* if and only if it is *non-dominated* by any other solution in the entire search space, which means that any other solution Y that improves one of the objective functions worsens other objectives. The set of all the non-dominated solutions is called the *Pareto-optimal set*, and the corresponding *objective vectors* form the *Pareto frontier*.

Multi-objective genetic algorithms, namely MOGAs, are often applied to solve this problem. Yoo and Harman [4, 5] explore test case selection through an experimental study, employing the additional greedy algorithm, NSGA-II algorithm, and a variant of NSGA-II, called the vNSGA-II algorithm. The findings indicate that the additional greedy algorithm is more suitable for single-objective test case selection. In contrast, NSGA-II and vNSGA-II algorithms perform better in multi-objective test case selection.

They [4, 5] also considered three contrasting criteria for the selection: code coverage, execution time, and fault history information. Then, empirical results showed no clear winner between MOGAs and additional greedy [4], nor did a hybrid approach produce better results [5].

Later on, Panichella et al. [8] introduced DIV-GA, an algorithm that improves the NSGA-II algorithm by injecting diversity into the genetic algorithm, reducing the genetic drift phenomenon in NSGA-II and enhancing the effectiveness and efficiency of test case selection.

2.2 Quantum Optimization for Test Case Selection

The "quantum era" promises to impact program computation so that even NP-hard problems will be solvable.

Qubits, generally represented by the electron spin, can represent different variable values at the same time. While a bit can represent a zero or a one, a qubit can be in a state of either of the two classical values with a certain probability. The actual value of a qubit is then known with the process of measurement, where the qubit is collapsed to a classical bit; when this happens, the qubit must be reset to make it reusable as a qubit.

Quantum optimization algorithms, such as Grover's Algorithm [30], use quantum oracles to search for unknown spaces with linear complexity. Quantum environments, e.g., D-Wave Quantum Leap, optimize NP-hard problems through adiabatic quantum optimization, defining the problem as a quantum system and its energy as a Hamiltonian to find the lowest energy solution.

Quantum approximate optimization algorithms (QAOAs) showed great potential in solving optimization algorithms, exploiting the capabilities of both classical and quantum computations. Wang et al. [19] proposed IGDec-QAOA, a method that resolves the test suite optimization problem as a QAOA problem, reaching significant effectivity levels; as explained earlier, we decided to leave further comparisons with IGDec-QAOA as future work.

Resolving optimization problems is the main focus of quantum annealers, a special kind of quantum computer designed for this specific goal, which applies the quantum version of simulated annealing: *Quantum Annealing* (QA). While the gate model divides problems into a sequence of operations on the qubits, quantum annealing translates the problem into a quantum system of qubits to find the minimum energy configuration through a gradual quantum state transition.

The method takes the steps from the *Adiabatic Theorem [31][32], according to which, starting a quantum system from its ground state, which represents its minimum energy state, if the Hamiltonian of the system changes slowly enough, the system will remain in its ground state during the whole evolution process. It solves combinatorial optimization problems by starting from a trivial- or easy-to-compute ground state related to an initial Hamiltonian and then slowly evolving the system towards a final Hamiltonian representing the problem to solve; thus, its ground state would represent the solution to the problem.*

Quantum annealing formalizes combinatorial optimization problems as single-objective ones. In particular, D-Wave uses the QUBO (Quadratic Unconstrained Binary Optimization) model to represent the objective function to minimize (i.e., the final Hamiltonian). It allows evaluating multiple states simultaneously and finding optimal solutions more efficiently, especially in complex landscapes where classical methods may struggle [24]. This heuristic algorithm cannot guarantee finding the optimal solution; therefore, it uses multiple sampling processes to generate candidate solutions in a single execution.

The main reason behind the choice of quantum annealing instead of gate-based quantum algorithms is the limitations of the former in available qubits, which are insufficient to resolve complex optimization problems. In particular, the two quantum annealing approaches experimented with in this work, BootQA and SelectQA, use different strategies to cope with highly complex and qubit-demanding problems. While BootQA performs a custom local decomposition strategy, that is, bootstrapping sampling, to obtain subproblems that can be solved directly by the QPU of the D-Wave Advanced System [33], SelectQA relies on the D-Wave Hybrid Solver Service (HSS) [34]. HSS takes the original QUBO model Q of the problem to solve as input and runs different hybrid solvers running on both CPUs and QPUs in parallel. The hybrid solvers query the D-Wave's Advantage QPU with sub-instances of the initial Q using Query Modules (QMs). The hybrid solvers interpret the results as suggestions about promising areas of the search space.

Wang et al. [18] pioneered using quantum annealing for test suite optimization, proposing BootQA. This quantum annealing tool is designed to resolve the test suite optimization problem using quantum annealing and circumvent the physical limitations of currently available quantum annealers. It is an alternative approach whose first purpose is *minimizing the number of test cases, promoting at the same time other defined objectives.* BootQA answers two research challenges: (i) designing a brand new QUBO model for the test suite optimization problem and (ii) designing a new method of qubit optimization. Due to their limited number of available qubits, the currently available quantum annealers can resolve large-scale problems only relying on classical computations as hybrid solvers do, like the one used by SelectQA [34].

Therefore, BootQA applies *bootstrap sampling* to decompose the original problem into smaller sub-problems (see Figure 1). The idea is to build small sub-problems by sampling subsets of test cases from the original test suite, ensuring that each generated sub-problem is solvable using the available quantum annealers.

Starting from the original test suite TS, BootQA decomposes it into m smaller test suites of size n. Of course, n is chosen based on the limitations of QA hardware. The m subsets are sampled *randomly*, meaning the same test case can appear in different sampled subsets. So, to make BootQA cover an exhaustive percentage of the test cases (depicted by the hyperparameter β), m is empirically set large enough. The resulting m subsets are used as input for the sub-problem

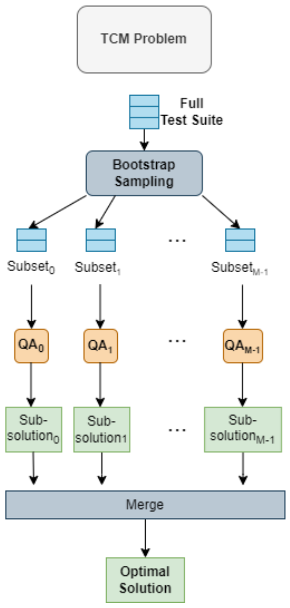


Fig. 1 BootQA [18] overview

QUBO formulation, then individually resolved by the annealing process, resulting in a sub-solution. For each sub-problem, an overall objective, which is an instance of the generic QUBO function, is constructed. Eventually, the different m sub-solutions derived from each sub-problem are merged into one single solution. In particular, each selected test case in each sub-problem is considered selected in the final solution.

Configuring BootQA to achieve the best subset of test cases could be challenging; therefore, we contacted the authors of the original paper [18] to ensure a correct configuration. Although the n parameter can be easily chosen, configuring the corresponding optimal m parameter is way more tedious. The configuration should assume values that allow the sampling process to obtain adequate coverage compared to the original test suite. The sampling should be conducted several times with different (m, n) pairs to select the optimal value, spending large amounts of execution time and possibly exceeding the time supplied by D-Wave. Furthermore, the random nature of sampling hinders direct control of the test case coverage, leading to the sampled sub-suites not representing the initial test suites and unstable solutions that do not represent good trade-offs between the objectives.

Table 1 Algorithm definitions

#	**Definition**
i	unique index identifying a test case
k	unique index representing a statement
Γ	the starting test suite
T_k	list of all test cases running the k-th statement
x_i	binary variable that specifies if the i-th test case is part of the solution
$cost(\tau_i)$	normalized execution cost of the i-th test
e_i	binary variable that indicates whether the i-th test case has detected errors in the past
f_i	the failure rate, i.e., the percentage of times a test case spotted a failure in the past
α	weight factor to modulate the contrast ratio between the objectives of the problem
P	penalty coefficient to regulate the weight of the constraints of the problem

3 SelectQA

This section introduces SelectQA, a method based on quantum annealing to solve the test case selection problem, as described in Section 2.1. To fully understand how the problem has been modeled, we provide the definitions of variables and functions necessary for its formulation in Table 1.

3.1 General QUBO Formulation

Formulating the test case selection problem as a QUBO problem involves a series of steps seamlessly integrating into a coherent framework. Initially, the primary objectives are set out, focusing on minimizing the execution cost of the test suite while maximizing its efficacy in detecting failures. We aim to compare the SelectQA approach with classical and quantum approaches, so we developed two different formulations: one coherent with the objectives chosen by Panichella et al. [8], the other coherent with the objectives chosen by Wang et al. [18]. In the three-objective version coherent with Panichella et al. [8], each test case is characterized by its execution cost, historical information about its ability to detect failures, and the statements it covers. In the two-objective version coherent with Wange et al. [18], each test case is characterized by its execution cost and the failure rate, i.e., the *percentage* of failures spotted by a test case based on the fault history. The two versions implement an algorithm that resolves the *Minimum Vertex Cover* problem. Furthermore, integrating the statement coverage constraint into the linear equation aligns with Serrano et al. [35].

3.1.1 The Three-Objective Version of SelectQA

The goal is to find the smallest subset of test cases that collectively covers all necessary program statements (those covered by the initial suite Γ), referred to as set S. Each test case is essentially a subset of S (in terms of the statements it covers), and the challenge is to identify the minimal collection of these subsets that efficiently covers the entire set S.

In the QUBO framework, this problem is expressed using binary variables (0 or 1) to represent the inclusion or exclusion of a test case in the final suite. The QUBO problem is then described by a Hamiltonian function or a Binary Quadratic Model (BQM), which incorporates the linear impacts of each test case and the quadratic terms representing interactions between different test cases, such as overlapping coverage of program statements. Following the methodologies proposed by Glover et al. [36], we transform the linear objectives of the test case selection problem into a quadratic form by creating a Hamiltonian that encapsulates the individual contributions of each test case and their interrelations. The formulated Hamiltonian is processed using quantum annealing, specifically the D-Wave system, designed for solving QUBO problems. This approach allows efficient solution space exploration to find the optimal subset of test cases. This method leverages quantum computing capabilities and promises more efficient solutions than classical algorithms, particularly for extensive and complex test suites.

The first goal of SelectQA consists of minimizing the overall execution cost of the resulting (sub-) test suite. Considering τ_i a generic test case and $cost(\tau_i)$ its corresponding **normalized** execution cost, the function that represents the first goal to minimize is formalized as follows:

$$\alpha \sum_{i=1}^{|\Gamma|} [x_i \cdot cost(\tau_i)] \tag{1}$$

The second objective of SelectQA consists of maximizing the overall fault coverage of the resulting (sub-) test suite. Here, having e_i indicating whether the $i-th$ test did detected a failure in the past, the function that represents the second goal to maximize is formalized below as:

$$(1-\alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \tag{2}$$

Since we want to minimize the overall objective Hamiltonian, the second objective function is converted to the following one to minimize:

$$-(1-\alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \tag{3}$$

Please note the presence of the α coefficient within the two target functions. This coefficient is a weight factor $(0 < \alpha < 1)$ to establish preference towards one goal rather than another using a weighted sum [37]. $\alpha = 0.5$ means that two objectives are equally important, whereas other values represent the more relevance of an objective over the other. The final function to be minimized consists of two parts and is formulated as follows:

$$H = \alpha \sum_{i=1}^{|\Gamma|} [x_i \cdot cost(\tau_i)] - (1-\alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \tag{4}$$

We introduce a critical constraint to maintain the coverage level of the initial test suite: each program statement executed in the original suite must be covered by at least one test case of the final selection. This constraint is essential to ensure that the final suite still comprehensively covers all necessary program statements despite reducing the number of test cases. This step allows maintaining the integrity and effectiveness of the test coverage by adding the following constraint:

$$\sum_{i \in T_k} (x_i) \geq 1 \tag{5}$$

The constraint states that at least one test case that executes the k statement must be selected.

The list of test cases that run the k-th statement is T_k. Since a program has several statements, we apply the constraint for each statement to cover and obtain the following constraints:

$$\sum_k \big(\sum_{i \in T_k} (x_i - 1)^2\big) \tag{6}$$

Given the previously described objectives and constraints, we construct a Hamiltonian expression to transform the linear test case selection problem into a QUBO problem. To this end, we add a penalty constant (P) [36], which balances the importance of constraints within the Hamiltonian. The use of penalties is crucial when dealing with problems that include additional constraints other than the one requiring variables to be binary. This kind of problem can be re-formulated as QUBOs leveraging a penalty coefficient in the objective function, which represents an alternative to the explicit use of separated constraints. For minimizing objective functions, penalties are equal to zero for acceptable solutions and equal to some positive value for unacceptable solutions. In other words, the optimizer itself, by the introduction of penalties, will search for the solution to avoid incurring the penalties. Empirically, setting the P penalty weight can be done using the Upper Bound strategy, which consists of setting P to be slightly higher than the maximum possible value achievable by the objective function. This step ensures that the penalty aligns with the application domain and significantly influences the solution process [38]. Hence, we have:

$$\begin{aligned} H = {} & \alpha \sum_{i=1}^{|\Gamma|} [x_i \cdot cost(\tau_i)] \\ & - (1-\alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \\ & + P \cdot \sum_k \left(\sum_{i \in T_k} (x_i - 1)^2 \right) \end{aligned} \tag{7}$$

The last part of the equation $(P \cdot \sum_k (\sum_{i \in T_k} (x_i - 1)^2))$, representing the constraints can be simplified as follows:

$$P \cdot \sum_k \big(\sum_{i,j \in T_k} (x_i^2 + 1^2 + 2x_i x_j - 2x_j)\big) \tag{8}$$

which can be further simplified as follows:

$$\sum_k \big(\sum_{i,j \in T_k} (-Px_i + 2Px_i x_j)\big) \tag{9}$$

The resulting BQM expression of the Hamiltonian (i.e., QUBO in this case) is the following:

$$\begin{aligned} H = {} & \alpha \sum_{i=1}^{|\Gamma|} x_i \cdot cost(\tau_i) \\ & - (1-\alpha) \sum_{i=1}^{|\Gamma|} (e_i \cdot x_i) \\ & + \sum_k \sum_{i,j \in T_k} (-Px_i + 2Px_i x_j) \end{aligned} \tag{10}$$

3.1.2 The Two-Objective Version of SelectQA

The procedure to obtain a two-objective formulation of the problem to compare it with the same dataset used by BootQA is the same. In this case, we have a two-objective linear problem where we want to minimize the final suite execution cost while maximizing its failure rate. Without repeating the previous procedure, we formulate the two objectives and combine them into a linear equation. We obtain:

$$\begin{aligned} H = {} & \alpha \sum_{i=1}^{|\Gamma|} x_i \cdot cost(\tau_i) \\ & - (1-\alpha) \sum_{i=1}^{|\Gamma|} (f_i \cdot x_i) \end{aligned} \tag{11}$$

Where the costs are normalized, and f_i is a percentage value representing the failure rate.

4 Empirical Evaluation

This section describes our goal, research questions, and research methods.

4.1 Goal and Research Questions

The *goal* is to evaluate the efficiency and effectiveness of the quantum-annealing-based test case selection method SelectQA and compare it to classical and quantum state-of-the-art approaches. The perspective is of researchers and practitioners: while the former are interested in improving state-of-the-art and classical computing techniques, the latter are interested in having a practically exploitable solution to their testing problems. We aim to answer the following research questions:

RQ$_1$: Is SelectQA more **effective** than traditional state-of-the-art methods and than the BootQA method?

RQ$_2$: Is SelectQA more **efficient** than traditional state-of-the-art methods and than the BootQA method?

Since the metrics used to assess the efficiency and effectiveness of the classical and quantum strategies differ, the analysis is split into two parts. First, we describe the research method followed to compare SelectQA to its classical counterparts and the results of this empirical analysis. Then, we discuss how we compared SelectQA to BootQA and the results of this second analysis.

4.2 Comparing SelectQA to Traditional TCS Algorithms

Table 2 Characteristics of the programs under study

Program	LOC	# TCs	Description
flex	10,459	567	Fast lexical analyzer
grep	10,068	808	Regular expression utility
gzip	5,680	215	Data compression program
sed	14,427	360	Non-interactive text editor

4.2.1 Study Context

One of the *goals* of this study is to evaluate the performance of SelectQA in resolving the TCS problem in terms of the three objectives depicted earlier: code coverage (in particular *statement coverage*), execution cost, and past faults coverage. The *study context* consists of four GNU open-source programs from the *software-artifact infrastructure repository* (SIR)[39]: *flex*, *grep*, *gzip* and *sed*. Table 2 reports the main characteristics of such programs. The choice of these programs is not random since they have been used in previous work, including the one proposing DIV-GA [8]. The following describes the test case selection criteria and how they have been gathered.

- *Statement Coverage.* Statement coverage represents how test cases execute source code statements. To extract these data, we used *gcov*, which can track the statements executed by each test case in C programs.
- *Execution Cost.* We did not rely on their execution time as external factors could influence it. Instead, the cost is calculated by counting the executed elementary instructions. This approach is consistent with previous work by Panichella et al. [8]. We used *gcov* to determine the execution frequency of each basic block (i.e., a linear section of code that is not branched and has only one entry and exit point). Block count is preferred over line count, as one line may contain multiple branches or function calls.
- *Past Faults Coverage.* The SIR provides versions of the programs, each featuring *injected* faults, specifying, through the use of a *fault matrix*, whether a given test case has detected errors in the past. This information is translated into a binary value associated with the corresponding test cases.

4.2.2 Experiment Configurations

The compared algorithms have been executed ten times to ensure correct empirical analysis of the results and mitigate the effects of the random nature of quantum algorithms, such as the annealing used by SelectQA. We compared:

- *SelectQA* proposed in this paper, which resolves a QUBO reformulation of the TCS problem leveraging quantum annealing;
- *DIV-GA* by Panichella et al. leverages diversity to generate more effective solutions [8];
- *Additional Greedy* by Yoo and Harman [4] unifies the three objectives into a single objective function to minimize and incrementally build a set of non-dominated solutions.

SelectQA and additional greedy have been implemented in Python, the former leveraging the *dwave* and *dimod* libraries. To compare the three algorithms, we implemented the QUBO formalization for the three-objectives TCS problem (the results and the code to replicate the experiments follow the instructions in [40]). Since quantum annealing is a single-objective algorithm, we built its solution incrementally following the same strategy used by additional greedy. So, starting from the selected test cases obtained by the annealing process, we incrementally produced a set of non-dominated solutions (sub-suites) [40]. DIV-GA has been implemented using *MATLAB R2024a Global Optimization Toolbox*, customizing the *gamultiobj* routine; the generation of the initial population has been performed using the *rowexch*, *hadamard* and *sortrows* routines. We set the DIV-GA parameters as described in the original work [8]. All implementations are available as part of our online appendix [40].

4.2.3 Evaluation Metrics

Effectiveness. To evaluate the quality of a multi-objective optimization algorithm, its yielded Pareto frontier should be compared to the actual one. Nevertheless, when resolving large problems like TCS, knowing the actual Pareto frontier *a priori* is impossible due to the impossibility of an exhaustive search. The only way is to perform an *a posteriori* evaluation, building, following the work by Panichella et al. [8], a hybrid frontier composed of all the non-dominated solutions between all the different frontiers obtained by each different algorithm for all the runs. Such a hybrid frontier is called *reference Pareto frontier* [4]. Let $P = \{P_1, ..., P_l\}$ be the set of l different Pareto frontiers obtained after all the experiment runs by all the evaluated algorithms, the Pareto frontier P_{ref} is defined as follows.

$$P_{ref} \subseteq \bigcup_{i=1}^{l} P_i \tag{12}$$

where $\forall p \in P_{ref} \nexists q \in P_{ref} : q > p$.

Given the reference frontier, we computed the *number of non-dominated solutions*, i.e., the number of non-dominated solutions found by an algorithm selected for the final reference frontier.

To ensure the empirical reliability of the results, we statistically analyzed the results to check whether the differences between the results obtained by the compared algorithms are significant. The results obtained over ten independent runs have been compared using *Mann-Whitney U test* [41]. Significant p-values mean that the null hypothesis, i.e., there is no statistically relevant difference between the effectiveness of the algorithms, has to be rejected in favor of the alternative one: statistically speaking, one of the two algorithms exhibits more non-dominated solutions selected by the reference Pareto frontier (we reject the null hypothesis for p-values < 0.05). Finally, we estimated the magnitude of the difference between the performance reported by the algorithms using the *Vargha-Delaney effect size* ($\hat{A}_{12}$) [42]. The effect size is interpreted as *small* in the range $(0.34, 0.44]$ or $[0.56, 0.64)$, *medium* in the range $(0.29, 0.34]$ or $[0.64, 0.71)$, and *large* in the range $[0, 0.29]$ or $[0.71, 1]$.
Efficiency. We analyzed the mean total run time of the algorithms over ten independent runs to compare their efficiency. DIV-GA and additional greedy have been executed on an Apple Macbook Air featuring an M1 chip and 16GB of RAM. The quantum annealing process was run on the D-Wave Leap Hybrid Solver `hybrid_binary_quadratic_model_version2` [34]. We used the D-Wave's "run-time" metric to obtain reliable results, reporting the total time needed for the machine to finish the operation. We statistically analyzed the total run times obtained over ten independent runs by each of the three algorithms leveraging the *Mann-Whitney U test.* We quantified the magnitudes of their differences using the *Vargha-Delaney effect size* ($\hat{A}_{12}$).

4.2.4 Results: RQ1 - Effectiveness

Table 3 reports the means and standard deviations of the size of the Pareto frontier and the number of non-dominated solutions for the three compared algorithms obtained by executing them ten times independently. The additional greedy algorithm always finds the larger set of solutions, but those of the other algorithms generally dominate those solutions. Still, additional greedy performed better than the other approaches for the flex program. SelectQA is the most effective approach, finding the highest number of

Table 3 Mean Pareto size and number of non-dominated solutions obtained on average by the algorithms in ten runs

Program	Method	Pareto Size		Non-Dom Solutions	
		Mean	St. Dev.	Mean	St. Dev.
flex	SelectQA	187.0	-	187.0	-
	DIV-GA	140.0	-	140.0	-
	Add. Greedy	**567.0**	-	**205.0**	-
	Additional Method	150.0	2.5	150.0	2.5
grep	SelectQA	225.5	0.5	**207.5**	0.52
	DIV-GA	70.0	-	70.0	-
	Add. Greedy	**802.0**	-	177.0	-
	Additional Method	180.0	3.0	180.0	3.0
gzip	SelectQA	86.3	0.8	41.3	0.8
	DIV-GA	105.0	-	**105.0**	-
	Add. Greedy	**199.0**	-	71.0	-
	Additional Method	95.0	1.5	95.0	1.5
sed	SelectQA	131.0	-	**131.0**	-
	DIV-GA	99.6	13.4	99.6	13.4
	Add. Greedy	**356.0**	-	85.0	-
	Additional Method	120.0	5.0	120.0	5.0

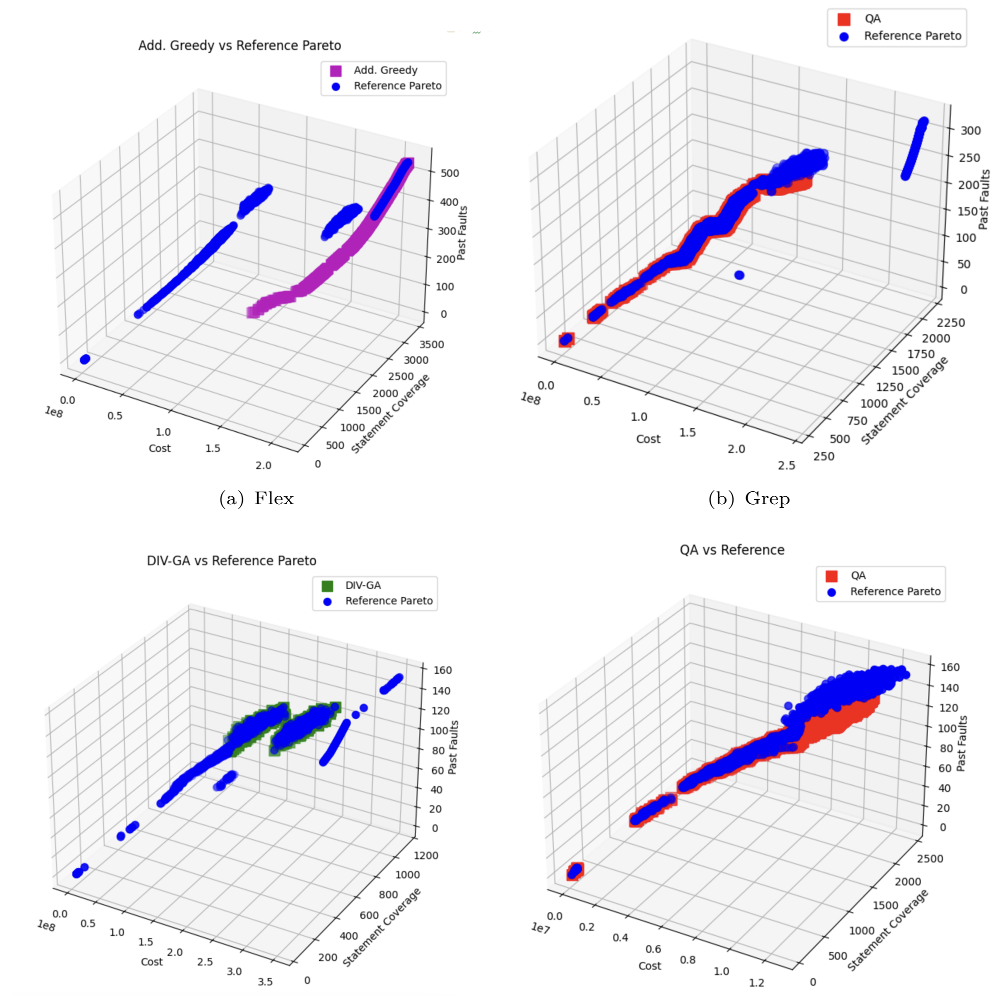
Table 4 Statistical comparisons between the algorithms in terms of the number of non-dominated solutions

Program	**Hypothesis**	**Non-Dom Solutions**	
		p-value	$\hat{\mathbf{A}}_{12}$
flex	SelectQA>DIV-GA	<**0.01**	1.0 (L)
	Add. Greedy>SelectQA	<**0.01**	1.0 (L)
	Add. Greedy>DIV-GA	<**0.01**	1.0 (L)
grep	SelectQA>DIV-GA	<**0.01**	1.0 (L)
	SelectQA>Add. Greedy	<**0.01**	1.0 (L)
	Add. Greedy>DIV-GA	<**0.01**	1.0 (L)
gzip	DIV-GA>SelectQA	<**0.01**	1.0 (L)
	DIV-GA>Add. Greedy	<**0.01**	1.0 (L)
	Add. Greedy>SelectQA	<**0.01**	1.0 (L)
sed	SelectQA>DIV-GA	<**0.01**	1.0 (L)
	SelectQA>Add. Greedy	<**0.01**	1.0 (L)
	DIV-GA>Add. Greedy	<**0.01**	0.9 (L)

non-dominated solutions among all the compared algorithms on two out of four programs. Concerning gzip, DIV-GA performs better than SelectQA and additional greedy. DIV-GA appears to be the most precise approach because all the solutions it finds are selected by the reference frontier.

To support the results, Table 4 reports the results of the Mann-Whitney U test and the Vargha-Delaney $\hat{A}_{12}$ effect size, obtained comparing the number of non-dominated solutions obtained by the three algorithms in the ten different runs. The test results confirm our previous observations. SelectQA performs statistically better than all the other algorithms in two of four programs, always with a large magnitude. In the case of gzip, DIV-GA performs statistically better than the other algorithms with large magnitude; the same is true for additional greedy regarding flex. Also, additional greedy performed better than SelectQA for gzip.

Figure 2 provides, for each program, a graphical comparison between the frontier obtained by a single execution of the best algorithm for that program and the corresponding reference Pareto frontier. The figure is consistent with the results reported in the previous section. Except for gzip and flex, SelectQA can always find the largest number of non-dominated solutions; we can see its frontier covering the larger part of the reference frontiers of grep and sed. DIV-GA is very precise: its solutions are always on the reference frontier, although generally, they are less than those obtained by SelectQA.



(a) Flex (b) Grep

(c) Gzip (d) Sed

Fig. 2 Additional Greedy performed the best with flex, while SelectQA with grep and sed, and DIV-GA with gzip.

Takeaway #1. SelectQA is the most effective approach in the number of non-dominated solutions, whereas DIV-GA remains unbeaten in the percentage of non-dominated solutions found.

4.2.5 Results RQ2 - Efficiency

Table 5 shows that SelectQA can resolve the problems constantly. Due to the use of singular value decomposition [8], DIV-GA is quite expensive and requires more computational time (compensated by the quality of its solutions). So, SelectQA is always more efficient than DIV-GA.

Table 5 Average execution times and standard deviations of the algorithms

	Add. Greedy		**DIV-GA**		**SelectQA**	
	Mean	**SD**	**Mean**	**SD**	**Mean**	**SD**
flex	8s	-	3m 39s	14s	**2.9s**	0.003s
grep	8s	-	1m 27s	5s	**2.9s**	0.003s
gzip	$<$ **1s**	-	19s	2s	2.9s	0.003s
sed	**1s**	-	1m 20s	8s	2.9s	0.003s

Additional greedy performed better than SelectQA in two cases: the gzip and sed programs. In all the other cases, SelectQA performs better than additional greedy. Please note that the additional greedy algorithm has a computational time of $O(|T| \cdot max|T_i|)$, in which $|T|$ is the size of the starting test suite and $max|T_i|$ represents the cardinality of the largest set of test cases able to reach the maximum coverage. The larger the system, the larger $max|T_i|$ will be, with high values leading to a higher number of iterations for the algorithm (because the maximum coverage to reach could be very high), meaning that applying additional greedy would be impractical for larger systems.

The statistical analysis confirmed the findings reported in Table 5, showing p-values always smaller than 0.01 with large effect sizes.

Takeaway #2. SelectQA has constant execution time despite the size of the program under test and outperforms DIV-GA in efficiency. Additional greedy could be impractical for large systems.

4.3 Comparing SelectQA to BootQA

4.3.1 Study Context

The *context* of this work consists of two real-world datasets, used in the previous work introducing BootQA [18]:*PaintControl* from ABB Robotics Norway [43] and *GSDTSR* [44] from Google. Both these datasets have the properties: "execution time" and "failure rate" (coherently with [18], we filtered out the test cases of both datasets with failure rates at zero, i.e., tests that never triggered failures during their history). We adapted SelectQA by applying the two-objective QUBO formulation presented in Section 3.1.2, where the two criteria are execution time and failure rate.

4.3.2 Experiment Configuration

The compared algorithms have been executed ten independent times to ensure correct empirical analysis of the results, especially since both tools rely on quantum algorithms.

Table 6 Best BootQA configuration for each dataset

Dataset	# TCs	n	m
PaintControl	89	30	6
GSDTSR	287	20	21

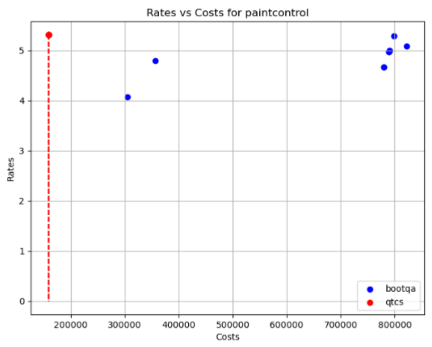
BootQA was directly cloned from its public repository in GitHub[45]. For both PaintControl and GSDTSR datasets, ten independent executions were conducted to empirically evaluate the best configurations of (n, m) parameters. Afterward, the algorithm was run ten independent times on the two datasets with the optimal configurations. Table 6 reports the best configurations found for GSDTSR and PaintControl, coherent with the base configuration [18]. As previously described, SelectQA has been implemented in Python, using the *dwave* and *dimod* libraries. All implementations are available as part of our online appendix [40].

4.3.3 Evaluation Metrics

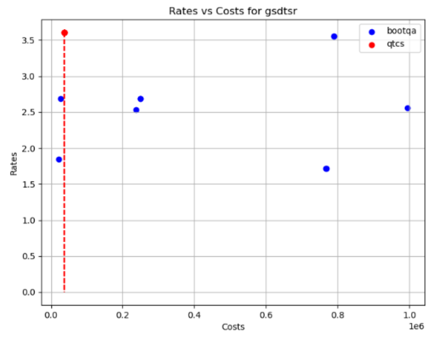
Effectiveness. Since both SelectQA and BootQA use a single-objective formulation to obtain a sub-optimal test suite, we compared the solutions by evaluating the execution times and failure rates of the test suites obtained by the two strategies. Coherently with the previous work on BootQA [18], we statistically analyzed the results obtained over ten independent executions by applying the *Mann-Whitney U test* [41] with a *significance level* set at 0.05. The null hypothesis represents a non-relevant difference between the two approaches. In contrast, if the null hypothesis is rejected, the magnitude of the difference is computed using the *Vargha-Delaney* effect size ($\hat{A}_{12}$) [42] (we reject the null hypothesis for p-values < 0.05). The effect size is interpreted as *small* in the range $(0.34, 0.44]$ or $[0.56, 0.64)$, *medium* in the range $(0.29, 0.34]$ or $[0.64, 0.71)$, and *large* in the range $[0, 0.29]$ or $[0.71, 1]$.
Efficiency. To compare the efficiency of BootQA and SelectQA, we analyzed their total run times. In particular, the former executes a local decomposition through bootstrap sampling (in this work, executed on a MacBook Air featuring an M1 Chip and 16GB of RAM) and directly runs the *Advanced System QPU*. The latter only relies on the `hybrid_binary_quadratic_model_version2` to handle highly complex optimization problems. For BootQA, we considered the total run times to be the sum of the bootstrap sampling process and Advanced System QPU execution time. In contrast, for SelectQA, we computed the total run time of the `hybrid_binary_quadratic_model_version2`.

The empirical reliability of the findings has been statistically validated by analyzing the distribution of the total run times, obtained over ten independent runs by each algorithm, using the *Mann-Whitney U test.* The magnitudes of the differences between the sequences have been quantified using the *Vargha-Delaney effect size* ($\hat{A}_{12}$).

4.3.4 Results: RQ1 - Effectiveness



(a) PaintControl



(b) GSDTSR

Fig. 3 Costs and failure rates comparisons

As seen in Figure 3, SelectQA could find the optimal solution to the problem in each run and for both datasets. The α parameter, balancing the weight of failure rate and execution time, can direct SelectQA to the solution representing the optimal trade-off between the two research objectives. It also allows engineers to decide whether one objective should be more relevant than another. BootQA could not find a solution that dominates SelectQA in terms of execution times and failure rates, while SelectQA found a solution that dominates those found by BootQA on 18 runs out of 20. In the remaining two cases on the GSDTSR dataset, BootQA could find two test suites with better execution times than those found by SelectQA but with worse failure rates; hence, BootQA could never dominate the solutions found by SelectQA.

Table 7 Statistical Comparisons between the Algorithms

Dataset	Hypothesis	p-value	$\hat{A}_{12}$
PaintControl	SelectQA<BootQA costs	**<0.01**	0.0(L)
	SelectQA>BootQA f rate	**<0.01**	1.0(L)
GSDTSR	SelectQA<BootQA costs	**0.02**	0.2(L)
	SelectQA>BootQA f rate	**<0.01**	1.0(L)

To understand the magnitude of the difference between the two approaches, we report in Table 7 the results of the Mann-Whitney U test and the Vargha and Delaney's $\hat{A}_{12}$ effect size, obtained by comparing the values of execution times and failure rates of the test suites obtained by the two approaches (over all the ten runs). The test results confirm the previous observations and point out a large magnitude of the difference between the two approaches in favor of SelectQA.

Takeaway #3. SelectQA outperforms BootQA in effectiveness, always finding the optimal trade-off solution and dominating BootQA solutions in most cases.

4.3.5 Results: RQ2 - Efficiency

Table 8 Average Execution Times of the Compared Algorithms

	SelectQA		BootQA	
	Mean	SD	Mean	SD
PaintControl	2.9s	0.003s	**0.029s**	<0.001s
GSDTSR	2.9s	0.003s	**0.030s**	<0.001s

As expected, Table 8 shows that BootQA is more efficient than SelectQA regarding the mean total run time because the problems to solve in BootQA are way smaller than in SelectQA. Also, the statistical analysis confirmed the finding, showing all the p-values as smaller than 0.01 with large 0 effect sizes.

Takeaway #4. BootQA outperforms SelectQA in total run time, showing the efficiency of a pure quantum solution.

5 Threats to Validity

This section discusses all aspects that could threaten the validity of the study conducted about the comparisons between SelectQA, DIV-GA, additional greedy, and BootQA.

- *Construct Validity.* The main threat in this regard concerns the correctness of the measures used to select tests: coverage, history of failures, cost of execution, and failure rate. We relied on open-source profiling and compilation tools (e.g., GNU gcc and gcov) and real-world case studies to limit this issue. DIV-GA has been implemented in MATLAB, while additional greedy and SelectQA have been implemented in Python. This difference could threaten validity due to the different code optimization mechanisms used by the two languages and the routines applied by the MATLAB environment to solve mathematical optimization problems. However, we performed this choice to replicate the DIV-GA base conditions and environment as presented in [8]. The choice of efficiency metric constitutes another threat to construct validity. We have chosen the total run time to compare SelectQA with the traditional and quantum algorithms. Such a metric provides a straightforward measure of how long the examined strategies take to solve the requested problem; however, this choice poses some significant challenges. The SelectQA total run time not only considers the time used by CPUs, like for its classic counterparts, but also the execution time of the Advantage QPU; hence, the total run time reported by SelectQA could be subject to considerable variability depending on QPU availability, resource queues, and classical-quantum communication overhead. In other words, the total run time of SelectQA could be subject to different bottlenecks due to various causes.
- *Internal Validity.* One of these threats is undoubtedly the random nature of quantum annealing algorithms. For this reason, the experiments were repeated ten times for each program under examination and then considering the means of the obtained results. The *tuning* of the P penalty parameter is also a factor that could undermine the internal validity of this job. Therefore, we applied a method well-known in literature [38] when choosing this value. The α parameter has been validated following repeated trials. We used the D-Wave default settings for other QA hyper-parameters configurations coherently with the original BootQA paper [18] and also asked the researchers themselves who worked on it about the parameters' tuning. Another threat to internal validity concerns the implementation of DIV-GA. The original version used the MATLAB *Global Optimization Toolbox* release R2011b, while the version used for this work is the MATLAB *Global Optimization Toolbox* release R2024a. So, the new version of DIV-GA could be subject to differences compared to its older version due to updated routines and different code optimization mechanisms. Also, the difference between the languages used to implement DIV-GA, additional greedy, and SelectQA could be seen as a threat to the internal validity due to the aforementioned fluctuations. This threat has been mitigated by executing different runs for each algorithm using the same datasets. Finally, another threat to internal validity is that using other sampling strategies for the (m, n) parameters of BootQA might result in better solutions. Our choice was to replicate the experiment conditions of the original BootQA work [18], so we did not experiment with different sampling strategies.
- *External Validity.* A dangerous threat to external validity corresponds to the impossibility of generalizing the obtained results, which is particularly true if the datasets are too small or distant from real-world scenarios. To mitigate this issue, the datasets chosen for the comparisons, extracted from reliable sources such as SIR, Google, and ABB Robotics Norway, have already been used for evaluating the approaches used as baselines in our study [4] [5] [6] [18] [8]. The chosen SIR programs represent real-world medium-scale and small-scale scenarios. On the other hand, the datasets used to compare SelectQA and BootQA represent real-world industrial medium-scale and small-scale datasets.

- *Conclusion Validity.* We interpreted the findings using appropriate statistical tests. To test the significance of the differences, we applied the (i) *Mann-Whitney U test* [41], while to estimate the magnitude and the effect size of the observed differences, we used the *Vargha-Delaney* [41] ($\hat{A}_{12}$) effect size. Conclusions are based only on statistically significant results.

6 Conclusions

This paper proposes SelectQA, a method for test case selection leveraging quantum annealing, and compares it to classical and quantum approaches. Regarding the classical approaches [4, 8], we found that it is the most effective approach in the number of non-dominated solutions, whereas DIV-GA [8], able to beat SelectQA in just one case, remains unbeaten in the percentage of non-dominated solutions found. Looking at the efficiency of the algorithms, SelectQA performs in constant time, demonstrating the superiority of quantum algorithms over classical ones due to the program size-independent execution time. Regarding BootQA [18], i.e., another method based on quantum computing, we found that the solutions provided by SelectQA dominate those offered by the other in most cases, whereas the opposite never happens. Nevertheless, BootQA is the most efficient algorithm due to the smaller size of problems deployed directly to the quantum annealer.

In future work, we plan to implement additional problem decomposition strategies with quantum algorithms to obtain more precise, stable, and efficient solutions. We also aim to consider other quantum algorithms in the experiments; in particular, we will implement a novel QAOA strategy to further compare classical, annealing, and QAOA strategies.

Acknowledgement

This work has been partially supported by the project "RECHARGE: monitoRing, dEtection, and CHaracterization of performAnce ReGrEssionss", grant P2022SELA7, under the PRIN 2022 PNRR MUR program funded by the EU - NGEU.

This work has been partially supported by the project "QUASAR: QUAntum software engineering for Secure, Affordable, and Reliable systems", grant 2022T2E39C, under the PRIN 2022 MUR program funded by the EU - NGEU.

Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or The European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible.

References

[1] Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: a survey. Software testing, verification and reliability **22**(2), 67–120 (2012)

[2] Rothermel, G., Harrold, M.J.: Empirical studies of a safe regression test selection technique. IEEE Transactions on Software Engineering **24**(6), 401–419 (1998)

[3] Perrouin, G., Oster, S., Sen, S., Klein, J., Baudry, B., Le Traon, Y.: Pairwise testing for software product lines: comparison of two approaches. Software Quality Journal **20**, 605–643 (2012)

[4] Yoo, S., Harman, M.: Pareto efficient multi-objective test case selection. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis, pp. 140–150 (2007)

[5] Yoo, S., Harman, M.: Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. Journal of Systems and Software **83**(4), 689–701 (2010)

[6] Yoo, S., Harman, M., Ur, S.: Highly scalable multi objective test suite minimisation using graphics cards. In: Search Based Software Engineering: Third International Symposium, SSBSE 2011, Szeged, Hungary,

September 10-12, 2011. Proceedings 3, pp. 219–236 (2011). Springer

[7] Wang, S., Ali, S., Gotlieb, A.: Minimizing test suites in software product lines using weight-based genetic algorithms. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 1493–1500 (2013)

[8] Panichella, A., Oliveto, R., Di Penta, M., De Lucia, A.: Improving multi-objective test case selection by injecting diversity in genetic algorithms. IEEE Transactions on Software Engineering **41**(4), 358–383 (2014)

[9] Arrieta, A., Wang, S., Markiegi, U., Arruabarrena, A., Etxeberria, L., Sagardui, G.: Pareto efficient multi-objective black-box test case selection for simulation-based testing. Information and Software Technology **114**, 137–154 (2019)

[10] Xue, Y., Li, Y.-F.: Multi-objective integer programming approaches for solving the multi-criteria test-suite minimization problem: Towards sound and complete solutions of a particular search-based software-engineering problem. ACM Transactions on Software Engineering and Methodology (TOSEM) **29**(3), 1–50 (2020)

[11] Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transactions on software engineering **33**(4), 225–237 (2007)

[12] Assunção, W.K.G., Colanzi, T.E., Vergilio, S.R., Pozo, A.: A multi-objective optimization approach for the integration and test order problem. Information Sciences **267**, 119–139 (2014)

[13] Epitropakis, M.G., Yoo, S., Harman, M., Burke, E.K.: Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In: Proceedings of the 2015 International Symposium on Software Testing and Analysis, pp. 234–245 (2015)

[14] Di Nucci, D., Panichella, A., Zaidman, A., De Lucia, A.: A test case prioritization genetic algorithm guided by the hypervolume indicator. IEEE Transactions on Software Engineering **46**(6), 674–696 (2018)

[15] Engström, E., Runeson, P., Skoglund, M.: A systematic review on regression test selection techniques. Information and Software Technology **52**(1), 14–30 (2010)

[16] Hoare, T., Milner, R.: Grand challenges for computing research. The Computer Journal **48**(1), 49–52 (2005)

[17] Knight, W.: Serious quantum computers are finally here. what are we going to do with them. MIT Technology Review. Retrieved on October **30**, 2018 (2018)

[18] Wang, X., Muqeet, A., Yue, T., Ali, S., Arcaini, P.: Test case minimization with quantum annealers. arXiv:2308.05505 [cs.SE] (2023)

[19] Wang, X., Ali, S., Yue, T., Arcaini, P.: Quantum approximate optimization algorithm for test case optimization. IEEE Transactions on Software Engineering (2024)

[20] D-Wave: D-wave environment. [Online] https://www.dwavesys.com (Accessed on: 14-11-2024)

[21] Farhi, E., Goldstone, J., Gutmann, S.: A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028 (2014)

[22] B. K. Chakrabart, A.D.: Colloquium: Quantum annealing e analog quantum computation. Reviews of Modern Physics, vol. 80, no. 3, p. 1061 (2008)

[23] Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse ising model. Physical Review E **58**(5), 5355 (1998)

[24] Suzuki, S.: A comparison of classical and quantum annealing dynamics. In: Journal of Physics: Conference Series, vol. 143, p. 012002 (2009). IOP Publishing

[25] Rothermel, G., Harrold, M.J.: Analyzing

regression test selection techniques. IEEE Transactions on software engineering **22**(8), 529–551 (1996)

[26] Fischer, K.F.: A test case selection method for the validation of software maintenance modifications (1977)

[27] Yau, S.S., Kishimoto, Z.: Method for revalidating modified programs in the maintenance phase. In: Proceedings-IEEE Computer Society's International Computer Software & Applications Conference, pp. 272–277 (1987). IEEE

[28] Rothermel, G., Harrold, M.J.: A safe, efficient algorithm for regression test selection. In: 1993 Conference on Software Maintenance, pp. 358–367 (1993). IEEE

[29] Bates, S., Horwitz, S.: Incremental program testing using program dependence graphs. In: Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 384–396 (1993)

[30] Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, pp. 212–219 (1996)

[31] Born, M., Fock, V.: Beweis des adiabatensatzes. Zeitschrift f¨ ur Physik, no. 6 (1928)

[32] Morita, S., Nishimori, H.: Mathematical foundation of quantum annealing. Journal of Mathematical Physics, vol. 49, no. 12 (2008)

[33] Catherine, M., Pau, F.: The d-wave advantage system: An overview. D-Wave, Tech. Rep., 2022. [Online] https://www.dwavesys.com/media/s3qbjp3s/14-1049a-a_the_d-wave_advantage_system_an_overview.pdf (Accessed on: 14-11-2024)

[34] McGeoch, C., Farré, P., Bernoudy, W.: D-wave hybrid solver service + advantage: Technology update. [Online] https://www.dwavesys.com/media/m2xbmlhs/14-1048a-a_d-wave_hybrid_solver_service_plus_advantage_technology_update.pdf (Accessed on: 14-11-2024)

[35] Serrano, M.A., Sánchez, L.E., Santos-Olmo, A., García-Rosado, D., Blanco, C., Barletta, V.S., Caivano, D., Fernández-Medina, E.: Minimizing incident response time in real-world scenarios using quantum computing. Software Quality Journal, 1–30 (2023)

[36] Fred Glover, R.H.e.Y.D. Gary Kochenberger: Quantum bridge analytics i: A tutorial on formulating and using qubo models. Ann Oper Res 314, 141–183 (2022)

[37] Steuer, R.E.: Multi-criteria optimization: Theory, computation, and application. John Wiley, New York (1986)

[38] Ayodele, M.: Penalty weights in qubo formulations: Permutation problems (2022)

[39] SIR: Software-artifact infrastructure repository. [Online] https://sir.csc.ncsu.edu/portal/index.php (Accessed on: 14-11-2024)

[40] Trovato, A.: Appendix. [Online] https://github.com/AntonioTrovato/SelectQA (Accessed on: 14-11-2024)

[41] Conover, W.J.: Practical nonparametric statistics. 3rd ed. New York, NY, USA: Wiley (1998)

[42] Vargha, A., Delaney, H.D.: A critique and improvement of the cl common language effect size statistics of mcgraw and wong. J. Educ. and Behav. Stat. 25(2), 101–132 (2000)

[43] Robotics, A.: Industrial robot supplier and manufacturer. [Online] http://new.abb.com/products/robotics (Accessed on: 14-11-2024)

[44] Google: Google code archive. [Online] https://code.google.com/archive/p/google-shared-dataset-of-test-suite-results/wikis/DataFields.wiki (Accessed on: 14-11-2024)

[45] Muqeet, A.: Bootqa. [Online] https://github.com/AsmarMuqeet/BootQA (Accessed on: 14-11-2024)