Test Case Minimization with Quantum Annealers

XINYI WANG and ASMAR MUQEET, Simula Research Laboratory, Oslo, Norway, and University of Oslo, Oslo, Norway
TAO YUE, Simula Research Laboratory, Oslo, Norway
SHAUKAT ALI, Simula Research Laboratory, Oslo, Norway, and Oslo Metropolitan University, Oslo, Norway
PAOLO ARCAINI, National Institute of Informatics, Tokyo, Japan

Quantum annealers are specialized quantum computers for solving combinatorial optimization problems with special quantum computing characteristics, e.g., superposition and entanglement. Theoretically, quantum annealers can outperform classic computers. However, current quantum annealers are constrained by a limited number of qubits and cannot demonstrate quantum advantages. Nonetheless, research is needed to develop novel mechanisms to formulate combinatorial optimization problems for quantum annealing (QA). However, QA applications in software engineering remain unexplored. Thus, we propose *BootQA*, the very first effort at solving test case minimization (TCM) problems on classical software with QA. We provide a novel TCM formulation for QA and utilize bootstrap sampling to optimize the qubit usage. We also implemented our TCM formulation in three other optimization processes: simulated annealing (SA), QA without problem decomposition, and QA with an existing D-Wave problem decomposition strategy, and conducted an empirical evaluation with three real-world TCM datasets. Results show that *BootQA* outperforms QA without problem decomposition and QA with the existing decomposition strategy regarding effectiveness. Moreover, *BootQA*'s effectiveness is similar to SA. Finally, *BootQA* has higher efficiency in terms of time when solving large TCM problems than the other three optimization processes.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; • **Computer systems organization → Quantum computing**;

Additional Key Words and Phrases: quantum annealer, test case minimization, D-Wave, quantum computing

ACM Reference format:
Xinyi Wang, Asmar Muqeet, Tao Yue, Shaukat Ali, and Paolo Arcaini. 2024. Test Case Minimization with Quantum Annealers. *ACM Trans. Softw. Eng. Methodol.* 34, 1, Article 5 (December 2024), 24 pages.
https://doi.org/10.1145/3680467

This work is supported by the Qu-Test (Project#299827) funded by the Research Council of Norway. X. Wang is supported by Simula's internal strategic project on quantum software engineering. S. Ali also acknowledges support from the Quantum Hub initiative (OsloMet). P. Arcaini is supported by Engineerable AI Techniques for Practical Applications of High-Quality Machine Learning-Based Systems Project (Grant Number JPMJMI20B8), JST-Mirai; and by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST, Funding Reference number: 10.13039/501100009024 ERATO.
Authors' Contact Information: Xinyi Wang (corresponding author), Simula Research Laboratory, Oslo, Norway, and University of Oslo, Oslo, Norway; e-mail: xinyi@simula.no; Asmar Muqeet, Simula Research Laboratory, Oslo, Norway, and University of Oslo, Oslo, Norway; e-mail: asmar@simula.no; Tao Yue, Simula Research Laboratory, Oslo, Norway; e-mail: taoyue@gmail.com; Shaukat Ali, Simula Research Laboratory, Oslo, Norway, and Oslo Metropolitan University, Oslo, Norway; e-mail: shaukat@simula.no; Paolo Arcaini, National Institute of Informatics, Tokyo, Japan; e-mail: arcaini@nii.ac.jp

ACM 1557-7392/2024/12-ART5
https://doi.org/10.1145/3680467

1 Introduction

Quantum computing (QC) brings enormous computational power by performing computations based on quantum mechanical principles, including specialized features, e.g., superposition, entanglement, and quantum tunneling, to perform faster computations. It pledges to unravel complex computational problems more efficiently than classical computing [28]. One application of QC in the near future is solving combinatorial optimization problems. To this end, quantum annealers—a special kind of quantum computers—have been used to demonstrate solving combinatorial optimization problems with **quantum annealing (QA)** [24, 44, 49]. Nowadays, D-Wave Systems Inc.[1] has commercialized quantum annealers and continuously increases the number of quantum bits (qubits). Currently, the most powerful system from D-Wave is the *Advantage* system with more than 5,000 qubits. Theoretically, QA can solve optimization problems faster than state-of-the-art algorithms that run on classical computers [14]. However, due to the small scale of currently available quantum annealers, demonstrating their quantum advantage over the classical computers is practically impossible. Current research challenges include devising novel problem encoding mechanisms and optimally utilizing the limited number of available qubits. With this purpose, some research has already experimentally demonstrated the potential of using QA for solving combinatorial optimization problems [24, 44, 49].

Test optimization aims to improve the cost-effectiveness of software testing. Particularly, in the context of regression testing, test optimization can be applied to select and prioritize a subset of test cases for testing a new software version from the test suite used for testing previous software versions. Test optimization can involve various optimization objectives, such as minimizing the number of test cases to select to improve the test efficiency and maximizing the fault detection rate of the selected test cases to enhance the test effectiveness. However, the number of test cases typically available for complex industrial systems is extremely large. An exhaustive search is impossible even with powerful classical computers. Even a non-exhaustive search (e.g., with heuristics-based algorithms [6, 40, 53, 56], and with machine learning algorithms [42]) is limited to the available budget (e.g., time) and can explore a small subset of all possible solutions. Thus, we hypothesize that, with currently available (limited) hardware resources, quantum computers can reach at least similar performance of solving one of the test optimization problems, namely **test case minimization (TCM)** problems, compared to solving them on classical computers but with less time. There have been efforts to apply quantum algorithms to solve classical software testing problems [22, 35, 57]. However, those quantum algorithms are implemented for gate-based quantum computers, whose current sizes are still small (e.g., a maximum of 133 qubits are currently available in an IBM's Quantum Computer[2]), while the number of qubits available on the current D-Wave quantum computer has reached 5,000, which is more feasible for solving large-scale combinatorial optimization problems. Considering that the application of QA to TCM problems is straightforward since QA is specifically designed to solve combinatorial optimization problems (including TCM problems), and the fact that QA has not been investigated for solving optimization problems in software engineering, in this article, we explore the use of QA to solve TCM problems.

In this article, we aim to address the TCM problem for classical software with QA, which aims to find a minimum number of test cases for execution [42]. However, two main research challenges exist. First, considering that QA solves an optimization problem by formulating it as an *Ising Model* or a **quadratic unconstrained binary optimization (QUBO)** model, we, hence, need a novel formulation for TCM. Second, current quantum annealers have a limited number of qubits.

[1] https://www.dwavesys.com/
[2] https://quantum.ibm.com/services/resources

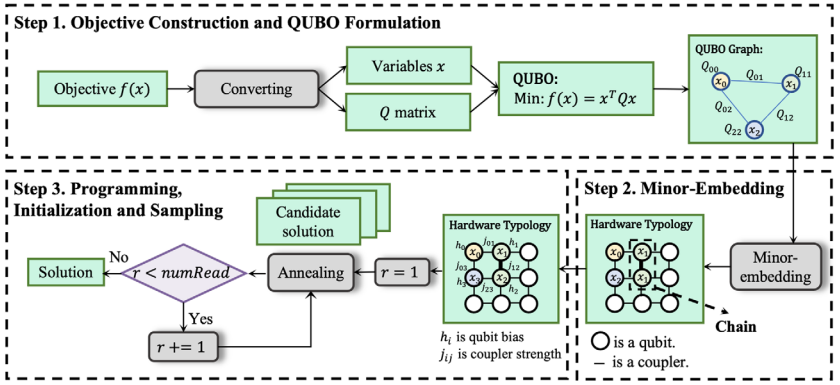


Fig. 1. Workflow of QA.

Therefore, they cannot be directly employed to solve complex TCM problems with large numbers of test cases, and new methods are needed to optimize the use of qubits.

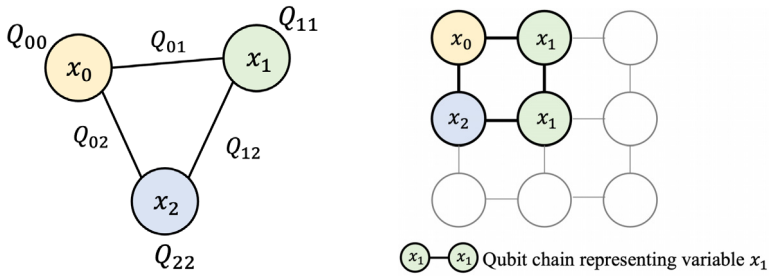
In this article, we propose *BootQA* to address the above two challenges. In particular, first, we propose a novel and generic QUBO formulation for TCM problems. Second, to optimize the use of qubits to solve large TCM problems with the current limited QC resources available, we introduce *bootstrap sampling* [31] to the QA process. To investigate the cost-effectiveness of *BootQA*, we also implemented our QUBO formulation in three other optimization processes: classical **simulated annealing (SA)**, QA without sub-problem decomposition, and QA with an existing decomposition strategy of D-Wave. We perform an empirical evaluation on three real-world datasets and evaluate the four optimization processes regarding their effectiveness and time efficiency in solving the three TCM problems. Results show that *BootQA* exhibits similar effectiveness with *SA* and outperforms the other two **quantum processing unit (QPU)**-based processes. Moreover, *BootQA* shows the highest time efficiency among all approaches, especially for large-scale TCM problems. We provide the experiment results and implementations in the online repository: https://zenodo.org/records/13273345.

In the rest of the article, Section 2 introduces the background, and Section 3 discusses the related work. Section 4 presents the proposed TCM approach with QA. Section 5 presents the empirical evaluation, and Section 6 presents the evaluation results, followed by discussions in Section 7. We conclude the article in Section 8.

2 Background

QA is based on the adiabatic theorem [39], stating that if a quantum system is initialized in a *ground state* (i.e., the lowest energy state) and the *Hamiltonian*.[3] i.e., the sum of both the kinetic and potential energy of the system. of the system changes sufficiently slowly, the quantum system will remain in the ground state during the system evolution. The adiabatic theorem can solve optimization problems by preparing a quantum system in a ground state of an easy-to-implement initial Hamiltonian and slowly evolving the initial Hamiltonian to the final Hamiltonian, whose ground state encodes solutions that solve the optimization problem. Below, we introduce the three main steps of solving an optimization problem with QA on a D-Wave QPU as described in [65]. The workflow of QA is composed of the three steps shown in Figure 1 and described in the following paragraphs.

[3]In quantum mechanics, the Hamiltonian of a quantum system specifies its total energy [14]



(a) QUBO graph (b) Working graph

Fig. 2. QUBO formalization for a three-variable problem.

Step 1: Objective Function and QUBO Formalization. A minimization objective function is defined as a mathematical expression representing the energy of the quantum system, which is converted into an Ising [25] or QUBO model [33] to be implemented on QA hardware. For D-Wave QA, QUBO is the standard model, which is a quadratic formulation with binary variables and formatted as a real-valued upper-diagonal weight matrix Q with its dimensions defined by the number of binary variables [33], defined as follows:

$$\min f(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} Q \boldsymbol{x} = \sum_i Q_{i,i} x_i + \sum_{i<j} Q_{i,j} x_i x_j \tag{1}$$

where $\boldsymbol{x}$ is the vector of binary decision variables, and $\boldsymbol{x}^{\mathrm{T}}$ is the transpose of $\boldsymbol{x}$. Diagonal terms $Q_{i,i}$ in Q are the linear coefficients, and off-diagonal terms $Q_{i,j}$ are quadratic coefficients. For example, for a three-variable (x_0, x_1, x_2) problem, the generic QUBO formulation is as follows:

$$\min f(\boldsymbol{x}) = \begin{bmatrix} x_0 \, x_1 \, x_2 \end{bmatrix} \begin{bmatrix} Q_{00} & Q_{01} & Q_{02} \\ 0 & Q_{11} & Q_{12} \\ 0 & 0 & Q_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \tag{2}$$

With Equation (1), we convert an optimization problem into a *QUBO graph* where each variable is represented as a node, and each non-zero off-diagonal term is represented as an edge between a pair of nodes connecting the corresponding variables, as shown, for example, in Figure 2(a).

Step 2: Minor-Embedding. We map a QUBO graph into a *working graph* describing how physical qubits are organized and connected in the QA hardware. The process is called *minor-embedding*, e.g., *Chimera topology* for the *D-Wave 2000Q* system and *Pegasus*[4] for the *Advantage* system.[5] In a working graph (e.g., Figure 2(b)), each node represents a *physical qubit*, and each edge connecting a pair of nodes (i.e., physical qubits) denotes a *coupler*. Ideally, each node in the QUBO graph (e.g., Figure 2(a)) should be mapped to a physical qubit, and each edge should be mapped to a coupler. However, since the *working graph* might not be fully connected, not all QUBO problems can be perfectly embedded into QA. For instance, to directly map the full-connected QUBO graph in Figure 2(a) onto the working graph in Figure 2(b), direct physical connections between any two pairs of three qubits representing three variables are needed, which is not the case for this working graph. To handle this, D-Wave introduced *chain*, which groups multiple physically connected qubits that can represent a node together (a variable in the QUBO formulation) of the QUBO graph. All the physical qubits in one chain act together as a single logical qubit to solve the optimization

[4]D-Wave System Documentation: https://docs.dwavesys.com/docs/latest/index.html
[5]The D-Wave Advantage System white paper: https://www.dwavesys.com/resources/white-paper/the-d-wave-advantage-system-an-overview/

problem. In this example, x_1 from the QUBO graph (Figure 2a) is mapped to two physical qubits in the working graph, i.e., a chain. As a result, each pair of variables is now connected in the working graph using this chain.

Minor embedding is time-consuming and requires specific algorithms to find suitable solutions [11]. A significant quantity of qubits and couplers is needed to embed some large and densely connected QUBO graphs. In this article, we used the algorithms implemented by D-Wave for minor embedding.

Step 3: Programming, Initialization, and Sampling. In the programming process, coefficients of the defined QUBO model are involved in defining the final Hamiltonian of the quantum system. Each linear coefficient is applied as *qubit bias*, while each quadratic coefficient is mapped as *coupler strength* in the QA hardware.

After programming, all qubits are initialized in equal superposition as the initial Hamiltonian of the system. Next, the quantum system evolves from the initial state (i.e., the initial Hamiltonian's ground state) to the final Hamiltonian's ground state, which encodes the objective function designed in Step 1. According to the adiabatic theorem, the system changes slowly enough to reduce the initial Hamiltonian's contribution and increase the final Hamiltonian's magnitude. During this process, the qubits' quantum dynamics slow down until a purely classical system is obtained. At this time, the magnitude of the initial Hamiltonian is decreased to 0, and the quantum system stays in the ground state of the final Hamiltonian. The qubits of the final state can be measured as the solution to the optimization problem.

Theoretically, by following the adiabatic theorem, quantum systems can guarantee the optimal solution. However, in practice, it is difficult to maintain the theoretical ground state of the quantum system. For example, the quantum system is very sensitive to the background noise of the quantum hardware and thermal fluctuations. Thus, the quantum system can easily get out of the ground state. Also, it is very hard to fulfill the condition of the adiabatic theorem, i.e., that the quantum systems should evolve "sufficiently slowly." In practice, the evolution time for a quantum system from the initial state to the final state is determined heuristically, making QA a non-deterministic algorithm.

Therefore, to reduce non-determinism, multiple sampling processes are required to generate multiple candidate solutions in an execution. The number of sampling processes is determined by the *number of reads* parameter in D-Wave QPU-API. Each sampling process has an annealing process and a read-out process. In the annealing process, the quantum system evolves from the initial to the final Hamiltonian in a time-dependent manner according to the Schrödinger equation. When the system reaches the final Hamiltonian's ground state, all qubits are read out, and the values represent a candidate solution. Generally, we consider the solution with the lowest energy as the optimal solution generated by QA.

3 Related Work

QC for Software Engineering. Researchers are increasingly exploring the potential of using quantum algorithms to tackle challenges in software engineering. To this end, eight categories of quantum algorithms have been identified that could potentially expedite software engineering tasks across different phases [36]. For instance, Harrow–Hassidim–Lloyd algorithm could enhance the speed of predicting code quality. Regarding software testing, one study has explored using quantum search algorithms, such as Grover's search and quantum counting, to reduce the computational complexity of dynamic testing [35]. Quantum search algorithms can also be applied for TCM through amplitude amplification [22]. Additionally, the quantum approximate optimization algorithm has been employed to tackle test case optimization problems [57]. However, these algorithms are designed for gate-based quantum computers; these are different from QA used in this article,

which executes on quantum annealers. The current gate-based quantum computers are small-scale (e.g., at most 133 qubits currently available in an IBM's Quantum Computer[6]). In contrast, the number of qubits available on the current D-Wave quantum computer has reached 5,000 in the D-Wave Advantage System, which is more feasible for solving large-scale combinatorial optimization problems.

QA for Optimization. Several QA applications in various domains have emerged recently, such as portfolio problem [47], machine learning [32], routing problems (e.g., traveling salesman [63], vehicle routing [52], control of traffic signal [23], process job scheduling [51]), and logistical network design [13]. Though QA has shown promising results in solving optimization problems in these domains, the potential for test case optimization remains unexplored, which is the focus of this article.

Test Case Optimization for Classical Software. Test case optimization is often classified into test case prioritization and TCM [4]. Test case prioritization aims to identify the most optimal ordering of test cases for execution, while TCM revolves around selecting a minimum subset of test cases that minimizes the overall test execution cost while maximizing the overall fault detection capability, and so on. Test case optimization has been extensively studied in the literature [9, 41, 64]. For instance, the authors of the survey in [40] identified and studied 90 test case optimization approaches and concluded that search-based (evolutionary algorithms, search algorithms) and clustering techniques are the most commonly used techniques. Another survey [42] analyzes 29 TCM and prioritization approaches with machine learning applied. The survey concludes that these approaches mainly employ supervised learning, reinforcement learning, and natural language processing-based methods. Current machine learning-based approaches in software development pipelines with continuous integration are unsuitable since they require the reconstruction of machine learning models from scratch from time to time. In contrast to existing works in the literature, we focus on using quantum annealers for TCM. Thus, our work is the first work on using quantum computers for solving TCM problems.

SA. SA is the classical counterpart of QA, it is a widely used heuristics-based optimization algorithm taking inspiration from a metal's heating and slow cooling to guide the search for the global optima. There have been various applications of *SA* and its variants in different domains [48] such as port throughput forecasting, transportation network problems, mobile robot navigation, and image processing. It has also been applied to solve software engineering problems, such as software defect estimation [26], software configuration tuning [10], combinatorial interaction testing [30], test case generation [54], and test case prioritization [68]. In this article, we apply *SA* on TCM problems to use it as a classical baseline approach to compare with *QA*.

Quantum Software Testing. In quantum software engineering, quantum software testing [18, 37, 38] is an active research sub-area. Several quantum software testing approaches have been proposed, such as search-based algorithms [60, 62], mutation testing [15, 17, 34], fuzzing [55], metamorphic testing [2], property-based testing [21, 45], coverage criteria [3, 59, 66], and combinatorial testing [58, 61], which all focus on generating test cases for testing quantum programs. In contrast, in this article, we propose *BootQA* to minimize test suites for testing classical software with quantum computers.

4 QA for TCM

This section, first, introduces the overview of the *BootQA* (Section 4.1), then defines the generic QUBO formulation (Section 4.2) illustrated with a running example.

[6] https://quantum.ibm.com/services/resources

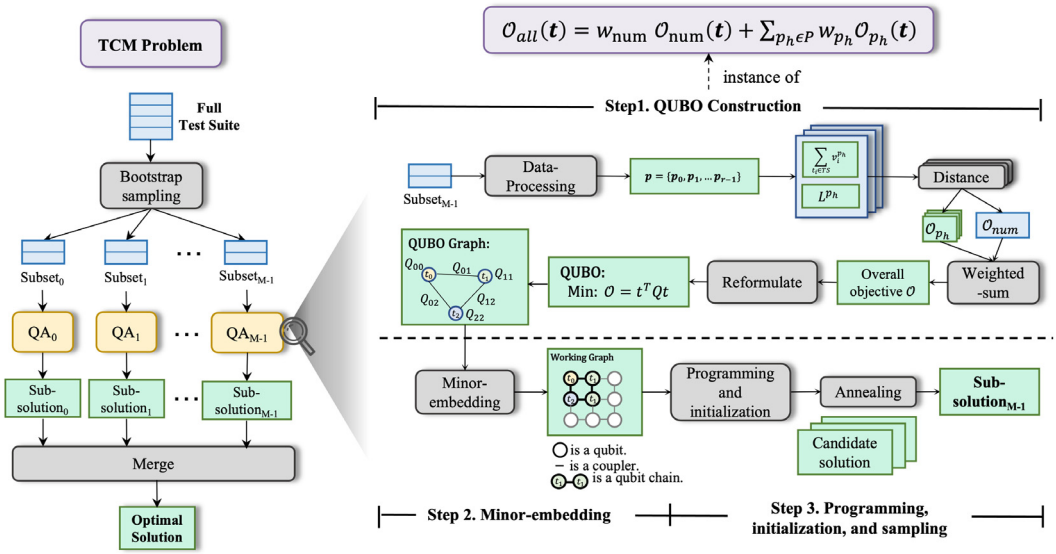


Fig. 3. Overview of *BootQA*.

4.1 Overview of *BootQA*

A QUBO formulation can be represented as a QUBO graph (see Section 2). If a QUBO graph has many nodes and connections among them, a substantial amount of qubits and couplers in the hardware are needed to embed the whole problem, which is not supported by today's QA hardware. To circumvent this issue, we use a bootstrap sampling strategy (Figure 3, left side) to decompose the original TCM problem into smaller sub-problems.

Bootstrap sampling randomly samples data into smaller subsets repeatedly with replacement, aiming to estimate a dataset's characteristics such that selected samples can effectively represent the whole dataset [20]. Specifically, in *BootQA*, bootstrap sampling creates small sub-problems by sampling independent subsets of test cases from the entire test suite to ensure that each sub-problem is solvable on the QA hardware. Note that each test case may appear in different subsets. Furthermore, since bootstrap sampling utilizes the random sampling technique, its execution time is negligible and hence ensures high efficiency.

Specifically, given a test suite TS, we decompose the original test suite into M smaller test suites of size N as shown in Figure 3. N is given by considering the maximum number of qubits and couplers supported by a given QA hardware. We randomly sample M subsets of TS of size N, i.e., $Subset_0$, $Subset_1$,..., $Subset_{M-1}$, repeatedly until a certain percentage of TS is covered. This coverage percentage is a hyper-parameter β of *BootQA*. These subsets are used as the input to formulate the sub-problems according to the QUBO formulation (Section 4.2). Each sub-problem is solved individually with QA, and, as a result, a corresponding sub-solution is produced.

For each sub-problem (Figure 3, right side), we construct an overall objective function based on test cases and their properties in the sub-problem, which is an instance of the generic QUBO formulation introduced in Section 4.2. We first extract the properties of test cases in each subset as optimization objectives. Second, we combine them into the overall objective function as a QUBO formulation (Step 1). With the QUBO graph converted from the formulation, we embed it into the hardware working graph (Step 2: minor-embedding). After the process of programming,

Table 1. Running Example

Test Case ID	Execution Time `et`(s)	Failure Rate `fr` (%)
t_0	10	80%
t_1	80	30%
t_2	50	40%
t_3	5	5%
t_4	100	90%

initialization, and sampling (Step 3), among all candidate solutions, we choose the solution with the lowest energy as the sub-solution of the sub-problem (see details in Section 2).

Eventually, for M sub-problems, we obtain M sub-solutions, each of which is a subset of the original test suite. Each selected test case in each sub-problem is considered as a selected test case in the final solution for the whole problem. In other words, we merge all the unique test cases selected in all sub-solutions to form the final solution for the whole problem.

4.2 QUBO Formulation for TCM

TCM selects the smallest possible subset of test cases out of the total set of test cases available, for the software under test, while satisfying all testing objectives as much as possible. Examples of objectives include minimizing the overall test execution time and maximizing the fault detection rate of the selected subset of test cases, whose corresponding values can be obtained based on the history executions. For TCM, the selection of each test case is represented as a binary variable t_i. If t_i takes the value of 1, it means that the ith test case is selected and 0 otherwise. Consequently, we represent the vector of s variables in the QUBO as $\boldsymbol{t} = [t_0, t_1, \ldots, t_{s-1}]$.

Each test case is characterized with r properties. Each property is associated with one testing objective. We represent the set of r properties as $P = \{p_0, p_1, \ldots, p_{r-1}\}$.

Example 1. To explain our approach, we use a small-scale running example composed of five test cases, shown in Table 1. Each test case has two properties: *execution time* (`et`) and *failure rate* (`fr`), shown as two columns in Table 1. Their values are based on the history of test executions. The test case execution time is the average time a test case takes across all its executions and the failure rate is the percentage of times a test case failed out of the total times it was executed in the continuous integration development context of the software. For example, test case t_0 has *execution time* of 10 seconds and *failure rate* of 80%. ◁

For the hth property, the vector of the values of all test cases is represented as $\boldsymbol{v}^{p_h} = [v_0^{p_h}, v_1^{p_h}, \ldots, v_{s-1}^{p_h}]$, where $v_i^{p_h}$ represents the hth property value of t_i. A TCM approach maximizes the overall effectiveness while minimizing the overall cost with the selected test cases. To this end, some properties shall be maximized (e.g., *failure rate* in our running example), and others shall be minimized (e.g., *execution time* in our running example). To treat all objectives equally in the overall objective function, we do the min-max normalization for properties whose ranges are not between 0 and 1.

For a property p_h, we aim to optimize the cumulative sum of property values for the selected test cases. Specifically, we maximize the sum of property values for all the selected test cases if p_h is an *effectiveness property*, and conversely, we minimize the sum if p_h is *cost property*. To do this, we define the objective function O_{p_h} (to be minimized) that computes the distance between the sum of values of the property for all selected test cases and the corresponding property's theoretical limit L^{p_h} with the Euclidean distance; L^{p_h} is the upper limit for an effectiveness property, and the lower

limit for a cost property. Formally:

$$O_{p_h}(\boldsymbol{t}) = (\boldsymbol{v}^{p_h} \cdot \boldsymbol{t}^{\mathrm{T}} - L^{p_h})^2, \tag{3}$$

where the sum of values of the property is calculated with matrix multiplication $\boldsymbol{v}^{p_h} \cdot \boldsymbol{t}^{\mathrm{T}}$, with $\boldsymbol{t}^{\mathrm{T}}$ being the transpose of $\boldsymbol{t}$. For an effectiveness property p_h, we calculate the theoretical limit as the sum of values of all the test cases for p_h. For a cost property, one possible theoretical limit is 0. However, depending on the context, other limits could be used.

Remark 1. Employing squares in calculating an objective is to give more penalty to a solution with the sum of the properties values of its selected test cases having a larger difference between the corresponding theoretical limit. ◁

With the total number of available test cases s, we can replace $\boldsymbol{v}^{p_h} \cdot \boldsymbol{t}^{\mathrm{T}}$ as $\sum_{i=0}^{s-1} v_i^{p_h} t_i$ in Equation (3) and expand the square to obtain

$$O_{p_h}(\boldsymbol{t}) = \left(\sum_{i=0}^{s-1} v_i^{p_h} t_i\right)^2 - 2L^{p_h} \sum_{i=0}^{s-1} v_i^{p_h} t_i + \left(L^{p_h}\right)^2. \tag{4}$$

To map the objective to the QUBO formulation (see Equation (1)), we need to avoid the squared terms. Note that for each variable $t_i \in 0, 1$, it shall hold that $t_i^2 = t_i$. This allows us to expand the first term in Equation (4) and incorporate the above formula as follows:

$$O_{p_h}(\boldsymbol{t}) = \sum_{i=0}^{s-1} \left({v_i^{p_h}}^2 - 2L^{p_h} v_i^{p_h}\right) t_i + 2 \sum_{i<j} v_i^{p_h} v_j^{p_h} \cdot t_i t_j + (L^{p_h})^2. \tag{5}$$

Example 2. For the objective of *failure rate* (i.e., the property `fr`) of our running example, we represent the vector of values as $\boldsymbol{v}^{\mathtt{fr}} = [v_0^{\mathtt{fr}}, v_1^{\mathtt{fr}}, v_2^{\mathtt{fr}}, v_3^{\mathtt{fr}}, v_4^{\mathtt{fr}}]$. The vector of five test cases is represented as $\boldsymbol{t} = [t_0, t_1, t_2, t_3, t_4]$. The upper theoretical limit is $L^{\mathtt{fr}}$. The objective formulation turns out to be

$$\begin{aligned} O_{\mathtt{fr}}(\boldsymbol{t}) &= (\boldsymbol{v}^{\mathtt{fr}} \cdot \boldsymbol{t}^{\mathrm{T}} - L^{\mathtt{fr}})^2 = \left(\sum_{i=0}^{4} v_i^{\mathtt{fr}} t_i - L^{\mathtt{fr}}\right)^2 \\ &= \sum_{i=0}^{4} ({v_i^{\mathtt{fr}}}^2 - 2L^{\mathtt{fr}} v_i^{\mathtt{fr}}) t_i + 2 \sum_{i<j} v_i^{\mathtt{fr}} v_j^{\mathtt{fr}} \cdot t_i t_j + (L^{\mathtt{fr}})^2. \end{aligned} \tag{6}$$

Since $\boldsymbol{v}^{\mathtt{fr}}$ =[0.8, 0.3, 0.4, 0.05, 0.9] and $L^{\mathtt{fr}} = 2.45$, the formula above is instantiated as

$$\begin{aligned} O_{\mathtt{fr}}(\boldsymbol{t}) = &-7.2t_0 - 2.85t_1 - 0.0487t_2 - 3.76t_3 - 8.01t_4 + 0.48t_0t_1 + 0.08t_0t_2 + 0.64t_0t_3 \\ &+ 1.44t_0t_4 + 0.03t_1t_2 + 0.24t_1t_3 + 0.54t_1t_4 + 0.04t_2t_3 + 0.09t_2t_4 + 0.72t_3t_4 + 24.01. \end{aligned} \tag{7}$$

Similarly, for the *execution time* (`et`) objective, we first normalize the property values to get vector $\boldsymbol{v}^{\mathtt{et}} = [v_0^{\mathtt{et}}, v_1^{\mathtt{et}}, v_2^{\mathtt{et}}, v_3^{\mathtt{et}}, v_4^{\mathtt{et}}] = [0.1, 0.8, 0.5, 0.05, 1.0]$. The objective function becomes

$$\begin{aligned} O_{\mathtt{et}}(\boldsymbol{t}) = &0.01t_0 + 0.64t_1 + 0.25t_2 + 0.003t_3 + t_4 + 0.16t_0t_1 + 0.1t_0t_2 + 0.01t_0t_3 \\ &+ 0.2t_0t_4 + 0.8t_1t_2 + 0.08t_1t_3 + 1.6t_1t_4 + 0.05t_2t_3 + t_2t_4 + 0.1t_3t_4. \end{aligned} \tag{8}$$

For objective `et`, we set its theoretical limit $L^{\mathtt{et}}$ to zero, implying that the sum of the selected test cases' execution time should be minimized.

As stated in Remark 1, we use squares in the objective calculation to penalize solutions with larger differences with the corresponding theoretical limits. For example, considering objective `et`, without taking the square, `et` can be written as $O_{\mathtt{et}}(\boldsymbol{t}) = 0.1t_0 + 0.8t_1 + 0.5t_2 + 0.05t_3 + t_4$. If we select all test cases (i.e., $\boldsymbol{t} = [1, 1, 1, 1, 1]$), the objective value is 2.45. By excluding t_4 (i.e., $\boldsymbol{t} = [1, 1, 1, 1, 0]$),

which has the highest execution time, the resulting objective value is 1.45 (so, a 41% reduction). The penalty incurred for t_4 in the et objective is only 1.00. However, with the squared formulation (8), removing t_4 incurs a penalty of 3.90, as selecting all test cases results in a value of 6.00, while excluding t_4 yields a value of 2.10 (so, a 65% reduction). Hence, incorporating the square into $O_{\mathtt{et}}(\boldsymbol{t})$ magnifies the impact of selecting t_4. ◁

A key objective of TCM is to minimize the number of test cases selected (we name it as num), i.e., the size of the selected test suite should be as small as possible. We add up the variable values in $\boldsymbol{t}(t_i \in \{0,1\}$ and $t_i = 1$ denoting that t_i is selected) to calculate num. To be consistent with the formalization of the other two properties, we represent the total number of test cases selected as a matrix multiplication: $\boldsymbol{v}^{\mathtt{num}} \cdot \boldsymbol{t}^{\mathrm{T}}$, where all values in $\boldsymbol{v}^{\mathtt{num}}$ are 1. Since we want to minimize the number of selected test cases, we consider its lower theoretical limit $L^{\mathtt{num}} = 0$. Consequently, the objective is represented as follows:

$$O_{\mathtt{num}}(\boldsymbol{t}) = (\boldsymbol{v}^{\mathtt{num}} \cdot \boldsymbol{t}^{\mathrm{T}} - L^{\mathtt{num}})^2 = \sum_{i=0}^{s-1} t_i + 2 \sum_{i<j}^{s-1} t_i t_j. \tag{9}$$

Example 3. In the running example, the objective function for num becomes

$$O_{\mathtt{num}}(\boldsymbol{t}) = \sum_{i=0}^{4} t_i + 2 \sum_{i<j}^{4} t_i t_j. \tag{10}$$

◁

The *overall objective function* O_{all} can be calculated based on the objectives corresponding to the properties and the objective of the number of selected test cases. We integrate all the objectives with the weighted-sum approach, where each objective is first normalized and then multiplied with a specific weight ($w_{\mathtt{num}}$ or w_{p_h} whose values are determined based on user preferences regarding the prioritization levels of all objectives) to reflect its priority in the TCM. For clarity, we do not show normalization in the rest of the calculation. The general formulation is shown below:

$$O_{all}(\boldsymbol{t}) = w_{\mathtt{num}} \cdot O_{\mathtt{num}}(\boldsymbol{t}) + \sum_{p_h \in P} w_{p_h} \cdot O_{p_h}(\boldsymbol{t}). \tag{11}$$

Example 4. In our running example, the overall objective is defined as follows:

$$O_{all}(\boldsymbol{t}) = w_{\mathtt{num}} O_{\mathtt{num}}(\boldsymbol{t}) + w_{\mathtt{et}} O_{\mathtt{et}}(\boldsymbol{t}) + w_{\mathtt{fr}} O_{\mathtt{fr}}(\boldsymbol{t}). \tag{12}$$

◁

We then transfer the overall objective function into a QUBO model. Since all constant terms are combined to represent the offset of the energy system, and these constant terms do not affect the optimization process, we do not need to consider them in the QUBO formulation. The general objective function is shown below (see Equation (1)):

$$\min O_{all}(\boldsymbol{t}) = \sum_{i=0}^{s-1} \left(w_{\mathtt{num}} + \sum_{p_h \in P} w_{p_h} \left({v_i^{p_h}}^2 - 2L^{p_h} v_i^{p_h} \right) \right) \cdot t_i + \sum_{i<j}^{s-1} \left(2w_{\mathtt{num}} + 2 \sum_{p_h \in P} w_{p_h} v_i^{p_h} v_j^{p_h} \right) \cdot t_i t_j, \tag{13}$$

where $w_{\mathtt{num}} + \sum_{p_h \in P} w_{p_h} ({v_i^{p_h}}^2 - 2L^{p_h} v_i^{p_h})$ represents the linear coefficients (the diagonal terms in the weight matrix Q) and $2w_{\mathtt{num}} + 2 \sum_{p_h \in P} w_{p_h} v_i^{p_h} v_j^{p_h}$ represents the quadratic coefficients (off-diagonal terms in Q).

Example 5. With the property set $P = \{\texttt{fr}, \texttt{et}\}$, the expanded objective function without the constant term is

$$O_{all}(\boldsymbol{t}) = \sum_{i=0}^{4} \left(w_{\texttt{num}} + \sum_{p_h \in \{\texttt{fr},\texttt{et}\}} w_{p_h} \left({v_i^{p_h}}^2 - 2L^{p_h} v_i^{p_h} \right) \right) \cdot t_i + \sum_{i<j}^{4} \left(2w_{\texttt{num}} + 2 \sum_{p_h \in \{\texttt{fr},\texttt{et}\}} w_{p_h} v_i^{p_h} v_j^{p_h} \right) \cdot t_i t_j \tag{14}$$

As an example, we set all three weight values as 0.33, and we obtain

$$\begin{aligned} O_{all}(\boldsymbol{t}) = &- 0.03t_0 + 0.0t_1 - 0.01t_2 + 0.01t_3 - 0.02t_4 + 0.04t_0t_1 + 0.04t_0t_2 + 0.03t_0t_3 \qquad (1) \\ &+ 0.05t_0t_4 + 0.04t_1t_2 + 0.03t_1t_3 + 0.06t_1t_4 + 0.03t_2t_3 + 0.05t_2t_4 + 0.03t_3t_4. \qquad (15) \end{aligned}$$

The objective function expressed using the Q matrix is

$$O_{all}(\boldsymbol{t}) = \boldsymbol{t}Q\boldsymbol{t}^{\mathrm{T}} = \begin{bmatrix} t_0 & t_1 & t_2 & t_3 & t_4 \end{bmatrix} \begin{bmatrix} -0.03 & 0.04 & 0.04 & 0.03 & 0.05 \\ 0 & 0 & 0.04 & 0.03 & 0.06 \\ 0 & 0 & -0.01 & 0.03 & 0.05 \\ 0 & 0 & 0 & 0.01 & 0.03 \\ 0 & 0 & 0 & 0 & -0.02 \end{bmatrix} \begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4. \end{bmatrix} \tag{16}$$

◁

5 Experiment Design

We first present the research questions in Section 5.1, followed by the experimental setup in Section 5.2, and evaluation metrics and statistical tests in Section 5.3.

5.1 Research Questions

RQ1 Does the proposed QUBO formulation solve TCM problems effectively?
This RQ examines whether our QUBO formulation is effective for TCM by checking the results of SA. If yes, we can use the same QUBO formulation for QA. This check is necessary, as we want to avoid any factor that may influence QA effectiveness that is not due to the formulation, such as hardware noise. Next, we compare *BootQA* with SA to determine if it performs similarly to SA in solving TCM to justify its use.

RQ2 How is the effectiveness of *BootQA* compared with the QA process of not employing any sub-problem decomposition?
This RQ checks if QA with bootstrap sampling can outperform QA without decomposing in solving a small-scale TCM problem. Note that solving large problems with it is practically infeasible.

RQ3 How does *BootQA* compare with an existing QA algorithm with a D-Wave built-in sub-problem decomposing technique?
This RQ checks whether *BootQA*'s bootstrapping sampling strategy helps it to perform better than an existing QA algorithm from D-Wave with its own subproblem decomposing process.

RQ4 How is the time efficiency of *BootQA* compared with the other three approaches?
This RQ aims to check the time cost of all approaches to obtain a comprehensive view of their overall time efficiency.

5.2 Experimental Setup and Execution

Datasets. We selected three real-world datasets to evaluate *BootQA*: *PaintControl* and *IOF/ROL* from ABB Robotics Norway[7] from [50] and *GSDTSR*[8] from Google. These datasets have been used for solving test case optimization problems in previous works [7, 50]. The available properties of the test cases in these datasets are "execution time" and "failure rate." We employed all available test cases in these datasets, except those whose failure rates are 0, as we need test cases in the original test suite *TS* that have the possibility to trigger failures. Eventually, we employed 89, 287, and 1,663 test cases from *PaintControl*, *GSDTSR*, *IOF/ROL*, respectively and formed the three datasets for our experiment to evaluate the performance of *BootQA* on various sizes of problems. Note that *PaintControl*, despite being a small-scale dataset, is still useful; first, it allows the performance assessment of *BootQA* on small-size problems, on which also classical algorithms could perform well; moreover, it allows the analysis of the performance of QA without decomposition, as done in RQ2.

Baselines. To evaluate *BootQA*, we also implement our TCM formulation in QA in the following three optimization processes and hence form three baselines. *(1) SA.* It is the classical counterpart of QA, so making it a suitable comparable baseline in our context. We use the dwave-neal framework from D-Wave as the SA baseline. It uses the QUBO formulation for optimization but with classical SA. *(2)* ***Vanilla QPU (VQ).*** We directly solve the TCM problems by running QA directly on D-Wave's QA hardware without decomposing. *(3)* ***Energy Impact Decomposer with QPU (EIDQ).*** We implement *EIDQ* as our baseline approach, which is tailored from the default hybrid QA algorithm from D-wave, Kerberos. This algorithm runs two classical branches (i.e., tabu search and (*SA*)) and a QPU branch in parallel and searches for an optimal solution by taking the best one from those produced by the three branches iteratively. The number of iterations is a hyper-parameter. In our experiment, we disabled the two classical branches since we already have *SA* for comparison. The QPU branch of Kerberos is interesting as it also combines a classical decomposing technique, i.e., Energy Impact Decomposer[9] and QA, which forms a good comparison baseline. For convenience, we name the modified Kerberos after the decomposing technique (i.e., Energy Impact Decomposer) and the QA as *EIDQ*. The decomposing technique applies search to find a sub-problem of a certain size that maximally contributes to the overall objective by calculating the energy impact of each variable on the fitness value.

Parameters. Each test case in each dataset has two properties: execution time and failure rate. For failure rate, we calculated the upper limit (L^{fr}) as the sum of the failure rates of all test cases in one dataset or one subset of it for *BootQA* since we aim to maximize the failure rate of the test suite. For execution time, we set the lowest limit (i.e., L^{et}) to 0 since we aim to minimize the execution time of the test suite. We use equal weights in O_{all}as an example (i.e., in practice, users can set weights of different objectives according to their own requirements). Moreover, we used the default settings in the D-Wave framework for hardware execution. For the three QPU-based approaches (i.e., *BootQA*, *EIDQ* and *VQ*), we set the number of reads (see Section 2) as 100 in each execution. For *SA*, we set the number of iterations as 100. The coverage rate β in our experiment is 90%.

We employed a range of sub-problem sizes N (defining the number of test cases in each sub-problem, see Section 4.2) to conduct a series of experiments with *BootQA*. For *GSDTSR* and *IOF/ROL*, we went from 10 to 160 with an increment of 10 to identify the best N. We stopped at 160 since our pilot study results show that the current QA hardware was unstable in finding embeddings

[7] http://new.abb.com/products/robotics

[8] https://code.google.com/archive/p/google-shared-dataset-of-test-suite-results/wikis/DataFields.wiki

[9] D-Wave Ocean Software Documentation, Hybrid Solvers: https://docs.ocean.dwavesys.com/en/stable/overview/hybrid.html

for more than 160 variables. For *PaintControl*, the total size of the dataset is 89; therefore, we use N from 10 to 80, with an increment of 10. For a fair comparison, we used the same number of variables (as for *BootQA*) as the maximum subproblem size of *EIDQ* in all iterations. We repeat experiments on each dataset 10 times for each subproblem size in *BootQA* and *EIDQ*. For *SA* and *VQ*, since there is no decomposition, we repeat experiments 10 times.

Execution Environment. We used the D-Wave Advantage system[10] as the QPU, and the classical part of the experiment was run on one CPU node in the Ex3[11] cluster with Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz.

5.3 Evaluation Metrics and Statistical Tests

5.3.1 Evaluation Metrics. We calculate the fitness value of each solution to evaluate the approaches' effectiveness and we name it as the *FV* metric (the lower, the better), which is related to the calculated objective function values. For *BootQA* and *EIDQ*, for each sub-problem size, *FV* for a TCM problem is calculated by taking the average value of the objective function (O_{all}) values of the 10 optimal solutions found in 10 repetitions. For *SA* and *VQ*, as there are no sub-problems, *FV* is calculated by taking the average value of the objective function (O_{all}) values of the 10 optimal solutions found for 10 repetitions. In addition, to compare *BootQA* with *SA* (RQ1) and *VQ* (RQ2), we evaluate their effectiveness with the *FV* metric for each subproblem size and also with the best sub-problem size. *BootQA* and *EIDQ* are compared (RQ3) by various subproblem sizes' effectiveness measured with *FV*. In RQ4, we evaluate the efficiency of *BootQA* in terms of the optimization time *exTime*. For *SA* and *VQ*, we calculate *exTime* as the sum of the classical CPU time for SA and QPU time for VQ. For *BootQA* and *EIDQ*, *exTime* is the sum of the decomposing time and the QPU time. For *BootQA* (or *EIDQ*), we calculate the bootstrap sampling time (or the decomposing time) and cumulative QPU portion time for all sub-problems. We used the best sub-problem sizes for both *BootQA* and *EIDQ* to evaluate the total time cost and the number of qubits when comparing with *SA* and *VQ*.

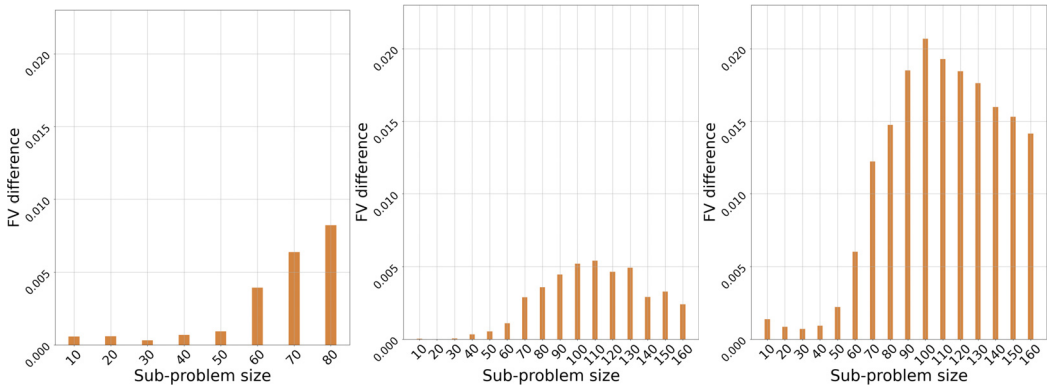
5.3.2 Statistical Tests. Following the guideline from [5], to establish the statistical significance of the results, we compare *BootQA* and *EIDQ* with the *FV* metrics (see Section 5.3.1) for all sub-problem sizes for each dataset using the Mann-Whitney U test and Vargha and Delaney's $\hat{A}_{12}$ effect size. We set the significance level as 0.05. The null hypothesis is that there is no significant difference between the two approaches. If the null hypothesis is not rejected, we consider the effectiveness of the two approaches to be equal. Otherwise, if the null hypothesis is rejected, we utilize Vargha and Delaney's $\hat{A}_{12}$ statistic as the effect size measure to quantify the magnitude of the difference between the two groups. If $\hat{A}_{12}$ is 0.5, the result is achieved by chance. If $\hat{A}_{12}$ is smaller than 0.5, then *BootQA* obtains lower values than *EIDQ* with a high probability and vice versa. For $\hat{A}_{12}$ statistics, by following [29], the effect size in the ranges $(0.34, 0.44]$ and $[0.56, 0.64)$ is interpreted as *Small*, in the ranges $(0.29, 0.34]$ and $[0.64, 0.71)$ is interpreted as *Medium*, and in the ranges $[0, 0.29]$ and $[0.71, 1]$ is interpreted as *Large*.

6 Experimental Results

In this section, we discuss experimental results to answer each RQ. The implementation of *BootQA* and all experiment results are provided in the online repository: https://github.com/AsmarMuqeet/BootQA.

[10]The D-Wave Advantage System white paper: https://www.dwavesys.com/resources/white-paper/the-d-wave-advantage-system-an-overview/

[11]EX3 is a national, experimental, heterogeneous computational cluster for researchers conducting experiments.



(a) *PaintControl* (b) *GSDTSR* (c) *IOF/ROL*

Fig. 4. RQ1—Differences between *BootQA* and *SA* in terms of *FV* for each sub-problem.

6.1 RQ1: Effectiveness of QUBO Formulation

We implement our QUBO formulation in *SA* to evaluate its effectiveness in solving three TCM problems of the three datasets. We obtain *FV* values of 1.19e-2, 1.74e-4, and 2.40e-2 for *PaintControl*, *GSDTSR*, and *IOF/ROL*, respectively, which are all very close to 0 (smaller is better), implying that our QUBO formulation effectively guides SA to find optimal solutions for all three TCM problems.

Encouraged by this, we then solve the TCM problems with *BootQA* with various sub-problem sizes (see Section 5.2). We calculated the *FV* value for each sub-problem size and the difference between the results of *BootQA* and those of *SA* is shown in Figure 4. We observed that for all three datasets, the *FV* values obtained by *BootQA* with all sub-problem sizes are close to those produced by *SA*; the average *FV* value differences are 2.71e-3, 2.63e-3, and 1.10e-2, for *PaintControl*, *GSDTSR*, and *IOF/ROL*, respectively. We also found that *BootQA* with lower sub-problem sizes (e.g., 10, 20, 30, 40) achieves closer performance to *SA*. When comparing *BootQA* configured with the best sub-problem sizes (30, 20, and 30 for *PaintControl*, *GSDTSR*, and *IOF/ROL*, respectively) with *SA*, the average *FV* differences are even more minor: 3.09e-4, 8.21e-6, and 7.04e-4 for *PaintControl*, *GSDTSR*, and *IOF/ROL*, respectively. Thus, we can conclude that the effectiveness of *BootQA* is practically similar to *SA*.

6.2 RQ2: Effectiveness of *BootQA* Compared to *VQ*

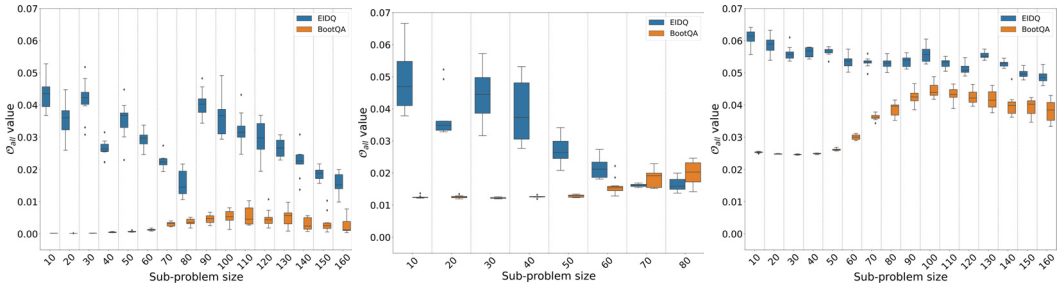
Large TCM problems cannot be directly solved on QA due to the limited number of qubits and couplers possible on the current QA hardware. For example, the overall QUBO graph scale of the two larger datasets (i.e., *GSDTSR* and *IOF/ROL*) exceeds the capacity of the current hardware. Thus, decomposing strategies are needed to fit a large TCM problem into the QA hardware.

To assess the effectiveness of using the bootstrap sampling strategy in QA to solve TCM problems, we compare the effectiveness of *BootQA* and *VQ* for solving a small TCM problem with *PaintControl*. Recall from Section 5 that we conduct experiments on *BootQA* configured with eight different sub-problem sizes for *PaintControl*. We used Mann-Whitney U statistical test and Vargha and Delaney's $\hat{A}_{12}$ statistics to compare O_{all}values produced by *BootQA* configured with each sub-problem size with those produced by *VQ* in 10 repetitions. Results in Table 2 show that when *BootQA* is configured with five out of the eight sub-problem sizes (i.e., 10, 20, 30 (the best sub-problem size), 40, and 50), the p-values are less than 0.05 with a large magnitude in terms of effect size in favor of *BootQA*. No significant difference can be observed when *BootQA* has the 60 and 70 sub-problem sizes. For

Table 2. RQ2—Comparison of *BootQA* and *VQ*

Sub-problem size	10	20	30	40	50	60	70	80
Result	✔$_{large}$	✔$_{large}$	✔$_{large}$	✔$_{large}$	✔$_{large}$	≡	≡	✗$_{large}$

✔$_{strength}$ (or ✗$_{strength}$) means that *BootQA* (or *VQ*) significantly outperforms *VQ* (or *BootQA*) with a given *strength* for the effect size (i.e., *small*, *medium*, *large*). ≡ means that there is no significant difference between *BootQA* and *VQ*.



(a) *GSDTSR* (b) *PaintControl* (c) *IOF/ROL*

Fig. 5. RQ3—Objective function (O_{all}) values produced by *BootQA* and *EIDQ* for each sub-problem.

the sub-problem size 80, *VQ* significantly outperforms *BootQA* with a large magnitude in terms of effect size. Thus, we can conclude that *BootQA*'s bootstrapping strategy with the best sub-problem size can potentially solve large TCM problems compared with implementing QA without decomposition.

6.3 RQ3: Effectiveness of *BootQA* Compared to *EIDQ*

Considering that both *BootQA* and *EIDQ* solve a TCM problem by splitting it into sub-problems, their sizes (i.e., the number of test cases in each sub-problem) might impact their performance. According to Section 5.2, we employ a range of sub-problem sizes for both *BootQA* and *EIDQ*. To compare them, for each sub-problem size, we analyze the objective function (O_{all}) values of the optimal solutions produced by *BootQA* and *EIDQ* in 10 repetitions. Results are shown in Figure 5. Comparing the three datasets, we can see that, in most cases, the objective function values of *BootQA* are lower than those of *EIDQ*, except for the cases with the 70 and 80 sub-problem sizes in *PaintControl*. Moreover, one can observe that for all three datasets, *BootQA* produced solutions with less variability among different sample sizes compared to *EIDQ*. *BootQA* follows a trend of having lower objective function values when the subproblem size is small and higher objective function values as the sub-problem size grows. In the case of *EIDQ*, no definite trend with respect to the sub-problem size can be observed. For instance, in *GSDTSR*, sometimes the objective function values are comparatively high even with small sample sizes like 10 and 30.

We find that *BootQA* consistently achieves lower objective function values with smaller sub-problem sizes (e.g., from 10 to 50) and larger objective function values for larger sub-problem sizes, as shown in Figure 5. One possible reason is that for a larger sub-problem, the length of the qubit chain will be longer, which affects the QA accuracy [1, 19]. In addition, for D-Wave QA, all coefficients in the QUBO formulation need to be scaled down to fit the fixed hardware bias and couplers determined by the physical properties of the qubits and their interconnections [19]. For

Table 3. RQ3—Best sub-problem sizes of *BootQA* and *EIDQ*

Dataset		*BootQA*		*EIDQ*	
Name	Size	N	M	N	*itr*
PaintControl	89	30	6	70	16
GSDTSR	287	20	21	160	7
IOF/ROL	1663	30	127	160	4

N is the sub-problem size; M denotes the number of subsets *BootQA* used for reaching 90% coverage; *itr*: the average number of iterations *EIDQ* used to produce the final solution.

a larger sub-problem (with more variables), the effective energy gap[12] becomes smaller. This is because the problem Hamiltonian gets more complex and has more energy levels, which results in a denser spectrum of energy levels and a smaller effective energy gap, consequently leading to lower optimization performance [19].

By observing the trend of objective function values generated by *BootQA* in Figure 5, we can observe a rise of the objective function values in *GSDTSR* and *IOF/ROL* at the sub-problem size being 110 and 100, respectively. The possible reason is that the scaling effect on different QUBO formulations varies, consequently affecting the results for different subproblems.

Overall, in most cases for all datasets, *BootQA* performs better than *EIDQ*. To establish the statistical significance of the results, we also evaluated the values of *FV* for all sub-problem sizes of *BootQA* and *EIDQ* with the Mann-Whitney U statistical test and Vargha and Delaney's $\hat{A}_{12}$ statistics. For all three datasets, the p-values are all less than 0.05, showing a significant difference between *BootQA* and *EIDQ*. For *GSDTSR* and *IOF/ROL*, the effect sizes are 0 (less than 0.5), indicating that *BootQA* is significantly better than *EIDQ* with a large magnitude for all subproblem sizes. For the *PaintControl* dataset, the effect size is 0.062, which is also less than 0.5, showing that *BootQA* significantly performs better than *EIDQ* with a large magnitude in most sub-problem sizes.

We also compare *BootQA* and *EIDQ* with their best sub-problem sizes for each dataset, regarding the *FV* values. Results are summarized in Table 3. *BootQA* gets the best results with the sub-problem sizes of $N = 20$ for *GSDTSR* and $N = 30$ for *IOF/ROL* and *PaintControl*. For *EIDQ*, the best settings are 70 for *PaintControl*, 160 for *GSDTSR* and *IOF/ROL*. The lowest *FV* produced by *BootQA* in *PaintControl*, *GSDTSR*, and *IOF/ROL* are 1.22e-2, 1.82e-4 and 2.46e-2, respectively, while that for *EIDQ* are 1.62e-2, 1.55e-2 and 4.87e-2. Comparing the two approaches, *BootQA* gets better performance in all datasets.

6.4 RQ4: Efficiency of *BootQA*

To compare the time efficiency, *BootQA* and *EIDQ* are set with their best sub-problem sizes. For *SA*, we calculate the computation time of the whole optimization process as *exTime*. For *BootQA*, *EIDQ*, and *VQ*, we employ the "QPU access time" metric [1] to measure the QPU portion in *exTime*, which is obtained via the D-Wave QPU-API.[13] Specifically, the QPU access time measures the total time spent on Step 3 (programming, initialization, and sampling) presented in Section 2. For *BootQA* and *EIDQ*, we use the time required for bootstrap sampling and Energy Impact Decomposer (see Section 5.2) as their decomposing time, respectively. Results are shown in Table 4.

[12]Effective energy gap is the difference of energy between the ground state and the first excited state of the quantum system [46].

[13]D-Wave System Documentation: https://docs.dwavesys.com/docs/latest/index.html

Table 4. RQ4—Time Performance (in Seconds) of the Four Approaches

Dataset	*SA*	*BootQA*			*EIDQ*			*VQ*	
	exTime	*exTime*		*Embedding*	*exTime*		*Embedding*	*exTime*	*Embedding*
		Decomposing	*QPU access*		*Decomposing*	*QPU access*			
PaintControl	0.440s ± 0.005	<0.001	0.171s ± 0.006	40.725s ± 6.019	1.542s ± 2.333	0.531s ± 0.411	641.678s ± 529.882	0.036s ± 0.001	76.091s ± 15.516
GSDTSR	2.811s ± 0.021	<0.001	0.890s ± 0.009	179.665s ± 10.211	1.438s ± 0.031	0.167s ± 0.001	1998.118s ± 385.059	-	-
IOF/ROL	86.333s ± 0.602	<0.001	3.656s ± 0.024	994.821s ± 10.245	70.726s ± 0.427	0.293s ± 0.002	3459.223s ± 730.282	-	-

Each value is shown in the format of *average* ± *standard deviation*. *exTime* values highlighted in the blue background are the smallest among all approaches in one dataset.

Regarding *exTime*, comparing all four approaches, for the *PaintControl* dataset, since *VQ* takes only one execution to solve the problem, its *exTime* is the shortest, while *EIDQ*'s *exTime* (the sum of the decomposing time and the QPU access time) is the longest. For *GSDTSR* and *IOF/ROL*, *BootQA*'s *exTime* is the shortest among all approaches (except for *VQ*, as it only applies to *PaintControl*).

When looking at each approach's time performance across the three datasets of various sizes, we can observe from Table 4 that *exTime* of *SA* grows significantly as the dataset size increases. Notably, the dataset size growth from *GSDTSR* to *IOF/ROL* is around 5.8 times (1, 663/287), whereas *SA*'s *exTime* surges around 30 times (86.333/2.811). For *EIDQ*, we observe that its *Decomposing* time takes a large part of *exTime*, which mainly depends on the number of iterations and the sub-problem size. On the other hand, the QPU access time of *EIDQ* is very short. Moreover, the standard deviation of its decomposing time varies more than the *BootQA*, which reflects the influence of the number of iterations on the decomposing time. For *BootQA*, the decomposing time of *BootQA* (i.e., bootstrap sampling time) is negligible for all datasets (i.e., less than 0.001 seconds), and the main time cost of *BootQA* is the QPU access time, which is affected by the problem size and the times of sampling M.

When comparing the time performance of all four approaches across all three datasets, *exTime* of *SA* and the decomposing time of *EIDQ* (i.e., the classical portion of *exTime*) grow dramatically with the increase of dataset size, which is, however, not the case for *BootQA*, where the decomposing time only increases from 0.171s to 3.656s. Thus, for large datasets, *SA* and some hybrid QA (e.g., *EIDQ*) may require considerable time to produce solutions. Comparatively, *BootQA* serves as a more time-efficient method for large TCM problems. However, the maximum speedup is achievable when the problem can be entirely mapped to a quantum computer, as demonstrated by *VQ*. In this case, *VQ* achieves the lowest *exTime* for *PaintControl*, as it does not necessitate decomposition. The efficiency achieved by *BootQA* and *VQ* indicates the advantage brought by quantum resources.

Regarding time for embedding, as discussed in Section 2, the general D-Wave QA process contains a minor-embedding phase, during which the heuristic-based D-Wave Advantage system[14] calculates an approximate mapping from a QUBO graph onto a quantum hardware working graph. This means that the embedding can differ in each run, even for the same QUBO graph on the same hardware. As shown in Table 4, the embedding time used by *BootQA* is the shortest as compared to *EIDQ* and *VQ*. Generally, the embedding time grows with the increase in the size of the QUBO graph [11]. However, we would like to mention that to compare with *SA*, though *BootQA* requires a significant amount of time for embedding, it is feasible to reuse the embedding across subproblems, as we will discuss in detail in Section 7. Therefore, the overall cost of applying *BootQA* can be further reduced. Optimizing the time and effectiveness of embedding is an active area of research [8, 67]. In the near future, the possibility of having a general embedding for fully connected QUBO graphs that can be used to run multiple problems in parallel will reduce the embedding cost drastically [43].

[14]The D-Wave Advantage System white paper: https://www.dwavesys.com/resources/white-paper/the-d-wave-advantage-system-an-overview/

6.5 Threats to Validity

Some threats may affect the validity of the proposed approach. We evaluated our approach on only three datasets due to the limitation of hardware resources. However, the three chosen datasets are industrial case studies from Google and ABB Robotics. Their sizes range from 89 to 1,663, which helps to evaluate the approaches with problems of various scales. We conducted our experiment with a single set of weights (i.e., equal weights) for each dataset, assuming that all optimization objectives have equal importance. Varying the weights may represent different degrees of importance for different objectives and thus will also result in different optimal solutions. To study whether our approach works for different weights would require an extensive empirical evaluation. Conducting such extensive empirical evaluation is constrained by the practical limitation on the QA hardware access, which affects the scale of the experiment. Evaluating *BootQA* on more datasets with more sets of weights to further evaluate its robustness, is therefore part of the future work, when QA hardware is more accessible.

Regarding the QA hyper-parameters configuration, we used the default settings from D-Wave in our experiment. Though we are aware that D-Wave QA solver instances have several hyper-parameters.[15] different settings of which may have an impact on the QA performance, empirically identifying the best settings of the hyperparameters is very costly, which we cannot afford for now.

Another threat to the empirical evaluation is the metrics used for answering RQs. We used the same fitness function for all approaches to evaluate their effectiveness. We use the experiment's average values of 10 repetitions to ensure accurate results. For calculating the time efficiency of the approaches, we used the standard metric, the QPU access time.[16] To establish the statistical significance of the results, we employed the Mann-Whitney U and Vargha-Daleney's $\hat{A}_{12}$ statistics to verify our findings. With the above, we provide a fair evaluation of all approaches and reduce the randomness during experiments.

7 Discussions

7.1 Enriching the QUBO Formulation with Dependencies

Our results showed that for all three datasets, *BootQA* performs better in terms of *FV* for smaller sub-problem size (i.e., N). This is because larger problem sizes require more chains and couplers in the working graph and scaling is required to fit the problem into the physical hardware which affects the performance of the algorithm. Though *BootQA* performs better with smaller N, it increases the number of M as all sub-problems together need to cover a required percentage of the dataset (e.g., 90%). A large M increases the overall execution time of QA, which might limit applying *BootQA* in cases where a solution is needed within a timing deadline. One solution is to optimize the QUBO formulation by considering dependencies among test cases (e.g., one test case needs to be executed before another). Our current QUBO formulation for the TCM problem makes a fully connected QUBO graph that cannot directly map to the QA hardware without using chains (see Section 2). Considering dependencies among test cases during the QUBO formulation could help make the QUBO graph sparsely connected, which can improve QA performance by requiring smaller chains and coupler strengths [1]. Also, with sparsely connected graphs, more variables can be mapped directly to physical hardware, which allows the use of large subproblem size N without reducing the effectiveness of the approach.

[15]D-Wave solver parameter list: https://docs.dwavesys.com/docs/latest/c_solver_parameters.html
[16]D-Wave System Documentation: https://docs.dwavesys.com/docs/latest/index.html

7.2 Constructing QUBO with Constraints in TCM

When addressing TCM problems, certain objectives may have constraints, such as setting a maximum time limit for test case execution, limiting the number of test cases to select, or imposing conditions on selecting or excluding particular test cases. Such constraints are often formulated as linear inequality constraints. In QUBO formulations, these constraints are often addressed by introducing additional variables called slack variables, which convert the inequality constraints into penalty functions that penalize specific objectives when the constraints are not satisfied [27]. However, incorporating slack variables requires additional qubits, thereby incurring additional costs. Since our current study mainly aims to demonstrate the applicability of QA for TCM problems, and considering that we have limited hardware resources, we do not consider TCM problems with constraints in this study. Regarding constraints for selecting specific test cases, in QUBO formulations, such constraints are managed by assigning a constant value 1 to variables representing test cases that must be selected due to the constraints. This introduces fixed terms in the objective functions. In the future, we plan to extend our study to accommodate TCM problems with constraints.

7.3 Reusing the Embedding across Sub-Problems

The minor-embedding phase is the most time-consuming part of the whole QA process, as we observed from the RQ4 results. A good embedding can significantly improve the accuracy and time performance of QA [1]. To solve a TCM problem with *BootQA*, the sub-problem size N is fixed, and given a fully connected QUBO graph, one can easily reuse the embedding of one subproblem to solve a new sub-problem. One possible solution to find a good embedding is to use search algorithms to find a good embedding of a fixed-size fully connected QUBO graph. Such an embedding can be used for both fully and sparsely connected QUBO graphs as long as the number of nodes in the graph remains the same. Reusing embeddings can significantly improve the time performance of QA, hence making it applicable in real settings even with the currently limited size of quantum hardware.

7.4 Limited by Hardware Resources

Current QA hardware has a limited number of qubits, which limits the size of optimization problems that can be solved with QA. However, from the results of RQ1, we observed that *BootQA* performs comparably to SA by dividing a bigger problem into smaller sub-problems. Table 5 shows the average number of physical qubits (*NQ*) used for solving the sub-problems with various sizes throughout 10 repetitions with *BootQA* and *EIDQ*.

As shown in Table 5, for each N, *NQ* used by both approaches is comparable. And it grows dramatically with the increase of N because the length of qubit chains grows with the increase in the number of variables. Specifically, for *VQ* with all 89 variables in *PaintControl*, the number of qubits used is around 1,000, which is enormous compared to the size of the problem. In contrast, *BootQA* can get relatively good performance with a small number of qubits; around 56 qubits were used when N was set 20, and 123 qubits were used for N set 30. For *EIDQ*, to get a good performance, as compared to *BootQA*, a lot more qubits are required. For instance, around 630 (3,407) qubits were needed for N being 70 (160). Thus, for solving TCM problems with a good performance by QA, the requirement of hardware for *BootQA* is not as high as that for other approaches, making *BootQA* a potentially more resource-efficient approach for solving these types of problems. In addition, we observe that with the current hardware, using more qubits does not guarantee good performance and the reasons are explained in RQ3 (Section 6.3).

However, in the future, advances in QC will increase the number of qubits and couplers, while reducing the noise in quantum computers. As a result, because of more couplers among qubits,

Table 5. Average Number of Physical Qubits (*NQ*) Used by *BootQA* and *EIDQ* for Solving all TCM Problems and Obtained *FV* Values

N	Dataset											
	PaintControl				*GSDTSR*				*IOF/ROL*			
	BootQA		*EIDQ*		*BootQA*		*EIDQ*		*BootQA*		*EIDQ*	
	NQ	*FV*	*NQ*	*FV*	*NQ*	*FV*	*NQ*	*FV*	*NQ*	*FV*	*NQ*	*FV*
10	17.3	0.0125	17.2	0.0490	17.2	0.0002	17.2	0.0432	17.2	0.0253	17.2	0.0610
20	**55.8**	**0.0125**	55.9	0.0376	**55.3**	**0.0001**	55.8	0.0355	**55.5**	**0.0248**	55.6	0.0587
30	**123.5**	**0.0122**	122.9	0.0443	**122.4**	**0.0002**	123.4	0.0417	**122.4**	**0.0247**	122.5	0.0560
40	212.1	0.0126	212.5	0.0389	213.7	0.0005	213.4	0.0264	214.4	0.0249	214.5	0.0564
50	326.5	0.0128	331.9	0.0272	328.3	0.0007	328.4	0.0353	330.3	0.0262	330.6	0.0566
60	468.2	0.0158	470.9	0.0216	465.0	0.0012	471.6	0.0294	467.1	0.0300	468.4	0.0534
70	621.1	0.0183	**629.8**	**0.0161**	623.8	0.0030	633.5	0.0226	627.5	0.0362	624.3	0.0532
80	817.2	0.0201	821.7	0.0164	827.3	0.0037	813.1	0.0156	823.3	0.0387	826.5	0.0530
90	-	-	-	-	1045.2	0.0046	1032.1	0.0404	1040.7	0.0425	1055.6	0.0536
100	-	-	-	-	1274.0	0.0053	1270.0	0.0365	1279.2	0.0446	1284.1	0.0558
110	-	-	-	-	1560.4	0.0055	1535.4	0.0323	1567.0	0.0432	1560.3	0.0530
120	-	-	-	-	1882.2	0.0048	1884.0	0.0298	1887.1	0.0424	1854.0	0.0512
130	-	-	-	-	2160.1	0.0051	2282.7	0.0265	2223.8	0.0416	2286.5	0.0553
140	-	-	-	-	2610.9	0.0030	2589.5	0.0224	2632.5	0.0399	2619.2	0.0528
150	-	-	-	-	3038.6	0.0034	3143.7	0.0186	3020.8	0.0393	3043.9	0.0497
160	-	-	-	-	3475.0	0.0025	**3406.8**	**0.0155**	3462.1	0.0381	**3406.7**	**0.0487**

NQ required and *FV* values produced by *BootQA* and EIDQ with the best sub-problem sizes are highlighted in bold.

the length of qubit chains will not grow dramatically with the increase of N. Thus, with shorter chains, more qubits, and less noise, *BootQA* may achieve much lower *FV* values with higher N than with current quantum annealers, thereby reducing the required M and enhancing the efficiency of *BootQA*.

7.5 Applying QUBO Formulations to Address Other Software Engineering Problems

In this article, we propose a QUBO formulation specialized for TCM problems, which can be generalized to address similar search-based software engineering problems, especially for subset selection problems, such as project planning problems in the management phase, and **next release problem (NRP)** in the Requirements/Specification phase. For instance, to address the NRP, a method needs to find a subset of tasks to implement in the next software release to minimize the cost and maximize customer satisfaction. Our QUBO formulation can be applied here, where each binary variable for TCM can represent each specific task in the NRP task list, the *cost property* can be considered as the cost of implementing the selected tasks, and the *effectiveness property* can be treated as the customer satisfaction score.

8 Conclusion and Future Work

Quantum annealers aim to solve combinatorial optimization problems, including those in software engineering, such as test optimization. In this article, we present the first work in the literature that provides a generic formulation of the TCM problem on classical software for QA. Moreover, given the fact that the current quantum annealers have a limited number of quantum bits (qubits), we propose a sampling method to optimize the use of physical qubits for TCM problems. We implemented our TCM formulation in four optimization processes: classical SA, QA without sampling (named *VQ*), and QA with two different sampling methods (*BootQA* we newly proposed in this article and *EIDQ* tailored from D-Wave's Kerberos). We evaluate them on three real-world datasets of various scales.

Results show that *BootQA* performs similarly as *SA*, and outperforms both *VQ* and *EIDQ* in terms of effectiveness. *BootQA* also shows higher time efficiency when solving large-scale TCM problems compared with the other three.

Our future work includes optimizing the QUBO formulation to further improve the efficiency of *BootQA*. We also plan to conduct more experiments with larger datasets and more weight sets to demonstrate the robustness of *BootQA*. Additionally, we plan to explore an adaptive weighting scheme for various objectives, such as based on uniform random adaptive weights assignment strategy [12]. Integrating the adaptive weight scheme with *BootQA* may enhance the effectiveness of solving TCM problems with QA. Moreover, we will compare QA with other classical search algorithms for solving TCM and other software engineering problems with an extensive empirical evaluation. We will also explore the potential of applying other quantum algorithms for TCM problems.

References

[1] Alastair A. Abbott, Cristian S. Calude, Michael J. Dinneen, and Richard Hua. 2019. A hybrid quantum-classical paradigm to mitigate embedding costs in quantum annealing. *International Journal of Quantum Information* 17, 05 (2019), 1950042. DOI: https://doi.org/10.1142/S0219749919500424

[2] Rui Abreu, João Paulo Fernandes, Luis Llana, and Guilherme Tavares. 2023. Metamorphic testing of oracle quantum programs. In *Proceedings of the 3rd International Workshop on Quantum Software Engineering (Q-SE '22)*. ACM, New York, NY, 16–23. DOI: https://doi.org/10.1145/3528230.3529189

[3] Shaukat Ali, Paolo Arcaini, Xinyi Wang, and Tao Yue. 2021. Assessing the effectiveness of input and output coverage criteria for testing quantum programs. In *Proceedings of the 2021 IEEE 14th International Conference on Software Testing, Validation and Verification (ICST)*, 13–23. DOI: https://doi.org/10.1109/ICST49551.2021.00014

[4] Mohammed Hazim Alkawaz and Abrahmi Silvarajoo. 2019. A survey on test case prioritization and optimization techniques in software regression testing. In *Proceedings of the 2019 IEEE 7th Conference on Systems, Process and Control (ICSPC)*, 59–64. DOI: https://doi.org/10.1109/ICSPC47137.2019.9068003

[5] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*. ACM, New York, NY, 1–10. DOI: https://doi.org/10.1145/1985793.1985795

[6] Aitor Arrieta, Shuai Wang, Urtzi Markiegi, Ainhoa Arruabarrena, Leire Etxeberria, and Goiuria Sagardui. 2019. Pareto efficient multi-objective black-box test case selection for simulation-based testing. *Information and Software Technology* 114 (2019), 137–154. DOI: https://doi.org/10.1016/j.infsof.2019.06.009

[7] Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel Briand. 2021. Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering* 48, 8 (2021), 2836–2856. DOI: https://doi.org/10.1109/TSE.2021.3070549

[8] Aaron Barbosa, Elijah Pelofske, Georg Hahn, and Hristo N. Djidjev. 2021. Optimizing embedding-related quantum annealing parameters for reducing hardware bias. In *Parallel Architectures, Algorithms and Programming*, Li Ning, Vincent Chau, and Francis Lau (Eds.). Springer Singapore, Singapore, 162–173. DOI: https://doi.org/10.48550/arXiv.2011.00719

[9] Tutku Çingil and Hasan Sözer. 2022. Black-box test case selection by relating code changes with previously fixed defects. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE '22)*. ACM, New York, NY, 30–39. DOI: https://doi.org/10.1145/3530019.3530023

[10] Tao Chen and Miqing Li. 2021. Multi-objectivizing software configuration tuning. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021)*. ACM, New York, NY, 453–465. DOI: https://doi.org/10.1145/3468264.3468555

[11] Vicky Choi. 2008. Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. *Quantum Information Processing* 7, 5 (2008), 193–209. DOI: https://doi.org/10.1007/s11128-008-0082-9

[12] Lucas RC de Farias, Pedro H. M. Braga, Hansenclever F. Bassani, and Aluizio F. R. Araújo. 2018. MOEA/D with uniformly randomly adaptive weights. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 641–648. DOI: https://doi.org/10.1145/3205455.3205648

[13] Yongcheng Ding, Xi Chen, Lucas Lamata, Enrique Solano, and Mikel Sanz. 2021. Implementation of a hybrid classical-quantum annealing algorithm for logistic network design. *SN Computer Science* 2 (2021), 1–9. DOI: https://doi.org/10.1007/s42979-021-00466-2

[14] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. 2000. Quantum computation by adiabatic evolution. arXiv: quant-ph/0001106. Retrieved from https://arxiv.org/abs/quant-ph/0001106

[15] Daniel Fortunato, José Campos, and Rui Abreu. 2022. Mutation testing of quantum programs: A case study with qiskit. *IEEE Transactions on Quantum Engineering* 3 (2022), 1–17. DOI: https://doi.org/10.1109/TQE.2022.3195061

[16] Daniel Fortunato, José Campos, and Rui Abreu. 2022. Mutation testing of quantum programs written in QISKit. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22)*. ACM, New York, NY, 358–359. DOI: https://doi.org/10.1145/3510454.3528649

[17] Daniel Fortunato, José Campos, and Rui Abreu. 2022. QMutPy: A mutation testing tool for quantum algorithms and applications in qiskit. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2022)*. ACM, New York, NY, 797–800. DOI: https://doi.org/10.1145/3533767.3543296

[18] Antonio García de la Barrera, Ignacio García-Rodríguez de Guzmán, Macario Polo, and Mario Piattini. 2023. Quantum software testing: State of the art. *Journal of Software: Evolution and Process* 35, 4 (2023), e2419. DOI: https://doi.org/10.1002/smr.2419

[19] Carlos D. Gonzalez Calaza, Dennis Willsch, and Kristel Michielsen. 2021. Garden optimization problems for benchmarking quantum annealers. *Quantum Information Processing* 20, 9 (Sep. 2021), 22 pages. 1570–0755. DOI: https://doi.org/10.1007/s11128-021-03226-6

[20] Tim Hesterberg. 2011. Bootstrap. *Wiley Interdisciplinary Reviews: Computational Statistics* 3, 6 (2011), 497–526.

[21] Shahin Honarvar, Mohammad Reza Mousavi, and Rajagopal Nagarajan. 2020. Property-based testing of quantum programs in Q#. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW' 20)*. ACM, New York, NY, 430–435. DOI: https://doi.org/10.1145/3387940.3391459

[22] Hager Hussein, Ahmed Younes, and Walid Abdelmoez. 2021. Quantum algorithm for solving the test suite minimization problem. *Cogent Engineering* 8, 1 (2021), 1882116. DOI: https://doi.org/10.1080/23311916.2021.1882116

[23] Daisuke Inoue, Akihisa Okada, Tadayoshi Matsumori, Kazuyuki Aihara, and Hiroaki Yoshida. 2021. Traffic signal optimization on a square lattice with quantum annealing. *Scientific Reports* 11, 1 (2021), 1–12. DOI: https://doi.org/10.1038/s41598-021-82740-0

[24] Daisuke Inoue and Hiroaki Yoshida. 2020. Model predictive control for finite input systems using the D-Wave quantum annealer. *Scientific Reports* 10, 1 (2020), 1591. DOI: https://doi.org/10.1038/s41598-020-58081-9

[25] Tadashi Kadowaki and Hidetoshi Nishimori. 1998. Quantum annealing in the transverse Ising model. *Physical Review E* 58, 5 (1998), 5355. DOI: https://doi.org/10.1103/PhysRevE.58.5355

[26] Sofian Kassaymeh, Mohamad Al-Laham, Mohammed Azmi Al-Betar, Mohammed Alweshah, Salwani Abdullah, and Sharif Naser Makhadmeh. 2022. Backpropagation Neural Network optimization and software defect estimation modelling using a hybrid Salp Swarm optimizer-based Simulated Annealing Algorithm. *Knowledge-Based Systems* 244 (2022), 108511. DOI: https://doi.org/10.1016/j.knosys.2022.108511

[27] Eric C Kerrigan and Jan M. Maciejowski. 2000. Soft constraints and exact penalty functions in model predictive control. In *Proceedings of the Control 2000 Conference*, 2319–2327.

[28] Alexei Yu Kitaev, Alexander Shen, and Mikhail N. Vyalyi. 2002. *Classical and quantum computation*. Number 47. American Mathematical Society USA.

[29] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. 2017. Robust Statistical methods for empirical software engineering. *Empirical Software Engineering*. 22, 2 (Apr. 2017), 579–630. 1382–3256. DOI: https://doi.org/10.1007/s10664-016-9437-5

[30] Tatsuya Konishi, Hideharu Kojima, Hiroyuki Nakagawa, and Tatsuhiro Tsuchiya. 2020. Using simulated annealing for locating array construction. *Information and Software Technology* 126 (2020), 106346, 0950–5849. DOI: https://doi.org/10.1016/j.infsof.2020.106346

[31] Anthony Kulesa, Martin Krzywinski, Paul Blainey, and Naomi Altman. 2015. Sampling distributions and the bootstrap. *Nature Methods* 12, 6 (Jun 2015), 477–478. 1548–7105. DOI: https://doi.org/10.1038/nmeth.3414

[32] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. 2021. Quantum computing methods for supervised learning. *Quantum Machine Intelligence* 3, 2 (2021), 23. DOI: https://doi.org/10.1007/s42484-021-00050-0

[33] Mark Lewis and Fred Glover. 2017. Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis. *Networks* 70, 2 (2017), 79–97. DOI: https://doi.org/10.1002/net.21751

[34] Eñaut Mendiluze, Shaukat Ali, Paolo Arcaini, and Tao Yue. 2022. Muskit: A mutation analysis tool for quantum software testing. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*. IEEE Press, 1266–1270. DOI: https://doi.org/10.1109/ASE51524.2021.9678563

[35] Andriy Miranskyy. 2022. Using quantum computers to speed up dynamic testing of software. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, 26–31. DOI: https://doi.org/10.1145/3549036.3562061

[36] Andriy Miranskyy, Mushahid Khan, Jean Paul Latyr Faye, and Udson C Mendes. 2022. Quantum computing for software engineering: Prospects. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering*, 22–25. DOI: https://doi.org/10.1145/3549036.3562060

[37] Andriy Miranskyy and Lei Zhang. 2019. On testing quantum programs. In *Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '19)*. IEEE Press, 57–60. DOI: https://doi.org/10.1109/ICSE-NIER.2019.00023

[38] Andriy Miranskyy, Lei Zhang, and Javad Doliskani. 2020. Is your quantum program bug-free? In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '20)*. ACM, New York, NY, USA, 29–32. DOI: https://doi.org/10.1145/3377816.3381731

[39] Satoshi Morita and Hidetoshi Nishimori. 2008. Mathematical foundation of quantum annealing. *Journal of Mathematical Physics* 49, 12 (2008). 00222488. DOI: https://doi.org/10.1063/1.2995837

[40] Rajendrani Mukherjee and K. Sridhar Patnaik. 2021. A survey on different approaches for software test case prioritization. *Journalof King Saud University. Computer and Information Sciences* 33, 9 (Nov. 2021), 1041–1054, 1319–1578. DOI: https://doi.org/10.1016/j.jksuci.2018.09.005

[41] TM Nithya and S Chitra. 2020. Soft computing-based semi-automated test case selection using gradient-based techniques. *Soft Computing* 24, 17 (2020), 12981–12987. DOI: https://doi.org/10.1007/s00500-020-04719-9

[42] Rongqi Pan, Mojtaba Bagherzadeh, Taher A. Ghaleb, and Lionel Briand. 2022. Test case selection and prioritization using machine learning: A systematic literature review. *Empirical Software Engineering* 27, 2 (Mar. 2022), 43 pages. 1382–3256. DOI: https://doi.org/10.1007/s10664-021-10066-6

[43] Elijah Pelofske, Georg Hahn, and Hristo N Djidjev. 2022. Parallel quantum annealing. *Scientific Reports* 12, 1 (2022), 1–11. DOI: https://doi.org/10.1038/s41598-022-08394-8

[44] Alejandro Perdomo-Ortiz, Joseph Fluegemann, Sriram Narasimhan, Rupak Biswas, and Vadim N Smelyanskiy. 2015. A quantum annealing approach for fault detection and diagnosis of graph-based systems. *The European Physical Journal Special Topics* 224 (2015), 131–148. DOI: https://doi.org/10.1140/epjst/e2015-02347-y

[45] Gabriel Pontolillo and Mohammad Reza Mousavi. 2023. A multi-lingual benchmark for property-based testing of quantum programs. In *Proceedings of the 3rd International Workshop on Quantum Software Engineering (Q-SE '22)*. ACM, New York, NY, USA, 1–7. DOI: https://doi.org/10.1145/3528230.3528395

[46] Atanu Rajak, Sei Suzuki, Amit Dutta, and Bikas K Chakrabarti. 2023. Quantum annealing: an overview. *Philosophical Transactions of the Royal Society A* 381, 2241 (2023), 20210417. DOI: https://doi.org/10.1098/rsta.2021.0417

[47] Gili Rosenberg, Poya Haghnegahdar, Phil Goddard, Peter Carr, Kesheng Wu, and Marcos López de Prado. 2016. Solving the optimal trading trajectory problem using a quantum annealer. *IEEE Journal of Selected Topics in Signal Processing* 10, 6 (2016), 1053–1060. DOI: https://doi.org/10.1109/JSTSP.2016.2574703

[48] Nazmul Siddique and Hojjat Adeli. 2016. Simulated annealing, its variants and engineering applications. *International Journal on Artificial Intelligence Tools* 25, 06 (2016), 1630001.

[49] Ilaria Siloi, Virginia Carnevali, Bibek Pokharel, Marco Fornari, and Rosa Di Felice. 2021. Investigating the Chinese postman problem on a quantum annealer. *Quantum Machine Intelligence* 3, 1 (2021), 3. DOI: https://doi.org/10.1007/s42484-020-00031-9

[50] Helge Spieker, Arnaud Gotlieb, Dusica Marijan, and Morten Mossige. 2017. Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. ACM, New York, NY, USA, 12–22. DOI: https://doi.org/10.1145/3092703.3092709

[51] Tobias Stollenwerk, Vincent Michaud, Elisabeth Lobe, Mathieu Picard, Achim Basermann, and Thierry Botter. 2021. Agile earth observation satellite scheduling with a quantum annealer. *IEEE Transactions on Aerospace Electronic Systems* 57, 5 (2021), 3520–3528. DOI: https://doi.org/10.1109/TAES.2021.3088490

[52] A. Syrichas and A. Crispin. 2017. Large-scale vehicle routing problems: Quantum annealing, tunings and results. *Computers & Operations Research* 87 (2017), 52–62. DOI: https://doi.org/10.1016/j.cor.2017.05.014

[53] Shanu Verma, Millie Pant, and Vaclav Snasel. 2021. A comprehensive review on NSGA-II for multi-objective combinatorial optimization problems. *IEEE Access* 9 (2021), 57757–57791. DOI: https://doi.org/10.1109/ACCESS.2021.3070634

[54] Hélène Waeselynck, Pascale Thévenod-Fosse, and Olfa Abdellatif-Kaddour. 2007. Simulated annealing applied to test generation: landscape characterization and stopping criteria. *Empirical Software Engineering* 12 (2007), 35–63. DOI: https://doi.org/10.1007/s10664-006-7551-5

[55] Jiyuan Wang, Fucheng Ma, and Yu Jiang. 2021b. Poster: Fuzz testing of quantum program. In *Proceedings of the 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 466–469. DOI: https://doi.org/10.1109/ICST49551.2021.00061

[56] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. 2015. Cost-Effective Test Suite Minimization in Product Lines Using Search Techniques. *Journal of Systems Software* 103, C (May 2015), 370–391. DOI: https://doi.org/10.1016/j.jss.2014.08.024

[57] Xinyi Wang, Shaukat Ali, Tao Yue, and Paolo Arcaini. 2023. Guess what quantum computing can do for test case optimization. *CoRR* abs/2312.15547 (2023). DOI: https://doi.org/10.48550/ARXIV.2312.15547

[58] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2021. Application of combinatorial testing to quantum programs. In *Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, 179–188. DOI: https://doi.org/10.1109/QRS54544.2021.00029

[59] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. Quito: A coverage-guided test generator for quantum programs. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*. IEEE Press, 1237–1241. DOI: https://doi.org/10.1109/ASE51524.2021.9678798

[60] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022b. QuSBT: Search-based testing of quantum programs. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22)*. ACM, New York, NY, USA, 173–177. DOI: https://doi.org/10.1145/3510454.3516839

[61] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2023b. QuCAT: A combinatorial testing tool for quantum software. In *Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2066–2069. DOI: https://doi.org/10.1109/ASE56229.2023.00062

[62] Xinyi Wang, Tongxuan Yu, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022c. Mutation-based test generation for quantum programs with multi-objective search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. ACM, New York, NY, USA, 1345–1353. DOI: https://doi.org/10.1145/3512290.3528869

[63] Richard H Warren. 2020. Solving the traveling salesman problem on a quantum annealer. *SN Applied Sciences* 2, 1 (2020), 75. DOI: https://doi.org/10.1007/s42452-019-1829-x

[64] Dharmveer Kumar Yadav and Sandip Dutta. 2020. Regression test case selection and prioritization for object oriented software. *Microsystem Technologies* 26 (2020), 1463–1477. DOI: https://doi.org/10.1007/s00542-019-04679-7

[65] Sheir Yarkoni, Elena Raponi, Thomas Bäck, and Sebastian Schmitt. 2022. Quantum annealing for industry applications: introduction and review. *Reports on Progress in Physics* 85, 10 (Sep. 2022), 104001. DOI: https://doi.org/10.1088/1361-6633/ac8c54

[66] Jiaming Ye, Shangzhou Xia, Fuyuan Zhang, Paolo Arcaini, Lei Ma, Jianjun Zhao, and Fuyuki Ishikawa. 2023. QuraTest: Integrating quantum specific features in quantum program testing. In *Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1149–1161. DOI: https://doi.org/10.1109/ASE56229.2023.00196

[67] Stefanie Zbinden, Andreas Bärtschi, Hristo Djidjev, and Stephan Eidenbenz. 2020. Embedding Algorithms for Quantum Annealers with Chimera and Pegasus Connection Topologies. In *High Performance Computing*, Ponnuswamy Sadayappan, Bradford L. Chamberlain, Guido Juckeland, and Hatem Ltaief (Eds.). Springer International Publishing, Cham, Switzerland, 187–206. DOI: https://doi.org/10.1007/978-3-030-50743-5_10

[68] Weixiang Zhang, Rui Dong, Bo Wei, Huiying Zhang, Sihong Wang, and Fengju Liu. 2022. Test Case Prioritization Based on Simulation Annealing Algorithm. In *Proceedings of the 8th International Conference on Computing and Artificial Intelligence (ICCAI '22)*. ACM, New York, NY, USA, 235–240. DOI: https://doi.org/10.1145/3532213.3532248

Received 14 August 2023; revised 26 June 2024; accepted 9 July 2024