Faster and Better Quantum Software Testing through Specification Reduction and Projective Measurements

NOAH H. OLDFIELD, Simula Research Laboratory and University of Oslo, Norway
CHRISTOPH LAABER, Simula Research Laboratory, Norway
TAO YUE, Simula Research Laboratory, Norway
SHAUKAT ALI, Simula Research Laboratory, Norway and Oslo Metropolitan University, Norway

Quantum computing (QC) promises polynomial and exponential speedups in many domains, such as unstructured search and prime number factoring. However, quantum programs yield probabilistic outputs from exponentially growing distributions and are vulnerable to quantum-specific faults. Existing quantum software testing (QST) approaches treat quantum superpositions as classical distributions. This leads to two major limitations when applied to quantum programs: (1) an exponentially growing sample space distribution and (2) failing to detect quantum-specific faults such as phase flips. To overcome these limitations, we introduce a QST approach, which applies a reduction algorithm to a quantum program specification. The reduced specification alleviates the limitations (1) by enabling faster sampling through quantum parallelism and (2) by performing projective measurements in the mixed Hadamard basis. Our evaluation of 143 quantum programs across four categories demonstrates significant improvements in test runtimes and fault detection with our reduction approach. Average test runtimes improved from 169.9s to 11.8s, with notable enhancements in programs with large circuit depths (383.1s to 33.4s) and large program specifications (464.8s to 7.7s). Furthermore, our approach increases mutation scores from 54.5% to 74.7%, effectively detecting phase flip faults that non-reduced specifications miss. These results underline our approach's importance to improve QST efficiency and effectiveness.

Additional Key Words and Phrases: Quantum computing, software testing, quantum program specification, projective measurements

1 Introduction

Quantum computers utilize the principles of quantum mechanics to perform computations at speeds unachievable by classical computers, opening unprecedented possibilities in fields such as quantum chemistry, optimization, machine learning, and cryptography [15, 18, 29, 43, 46, 61]. While the computational power of classical computers scales linearly for each added bit, quantum computational power scales exponentially for each added qubit. This exponential power comes from exploiting quantum superposition and entanglement in quantum algorithms, which one implements as quantum programs with a quantum software development kit such as IBM's Qiskit, Rigetti Forest or Google's Cirq [1, 10, 14]. Quantum programs, similar to classical programs, are also prone to bugs, therefore their correctness needs to be ensured. However, as quantum computing extends classical computing, quantum programs are found to contain not only classic software faults, but also quantum specific faults [4, 9, 48]. Therefore, quantum software testing (QST) plays a crucial role in the quantum software development cycle to find these faults, motivating the need for the development of software testing practices that meet the required standards of industrial and academic applications [2, 42, 76].

In the evolving landscape of QST, research has adapted and applied classical software testing approaches such as input-output coverage, differential, property-based, and metamorphic testing to quantum programs, attempting to incorporate quantum features [28, 41, 49, 71, 72]. These approaches validate quantum programs by statistically sampling the quantum program distribution in the computational basis and comparing a sample distribution with

Authors' addresses: Noah H. Oldfield, Simula Research Laboratory and University of Oslo, Oslo, Norway, noah@simula.no; Christoph Laaber, Simula Research Laboratory, Oslo, Norway, laaber@simula.no; Tao Yue, Simula Research Laboratory, Oslo, Norway, taoyue@gmail.com; Shaukat Ali, Simula Research Laboratory, Oslo, Norway and Oslo Metropolitan University, Oslo, Norway, shaukat@simula.no.

a theoretical distribution using a statistical test, such as the chi-square test. This theoretical distribution refers to different concepts depending on the approach. For instance, it is often formulated as program specification (PS) in input-output coverage [72] and mutation testing [41], is considered a property in property-based testing [28], or represents the expected statistical relationship between the distributions of a source and follow-up program in metamorphic testing [49].

We identify two challenges of the current QST approaches. First (1), current QST approaches require a considerable number of samples for scalable validation of a quantum program distribution. This scalability issue, however, is not yet noticeable, as current QST approaches have only been evaluated on tiny quantum programs having up to 12 qubits and a circuit depth of 30 [49, 72]. Scaling to 60 qubits introduces exponentially large search spaces, such as 2^{30} elements in Grover Searches, necessitating a high number of measurements [24, 52]. Similar measurement optimization challenges arise in other applications, such as quantum chemistry, where eigensolvers primarily optimize measurements of a single output value, specifically the expectation value of the Hamiltonian [20]. In contrast, quantum software testing frequently requires multiple sets of measurements to validate various outputs across numerous test cases. Thereby, due to the exponential scaling of quantum program distributions, efficient sampling is paramount for efficient QST. Second (2), quantum programs exhibit a multitude of bugs of which gate faults are a common source [4, 9, 48]. Current testing techniques often do not consider such quantum-specific bugs, as these techniques solely sample quantum probability distributions in the computational basis [46]. This happens due to phase flip faults manifesting in the program's final state vector as flipped signs in front of one or multiple basis states; thus, counting the basis states from measurements never reveals information about this flipped sign, as it is merely a mathematical artifact in a given basis. Only upon performing a projective measurement [46] in the Hadamard basis can we detect phase flip faults because then the fault can be detected as a flipped *bit*.

To tackle challenges (1) and (2), we propose a Greedy reduction-based approach to reduce the *ranks* of quantum program specifications by utilizing projective measurements in mixed Hadamard bases. According to the state vector postulate of quantum mechanics [46, 55], all possible information about a quantum system is contained in its state vector. Based on this, we define the PS as the non-faulty final state vector of the quantum program. Further, inspired by the Schmidt rank [46, 55], we define the rank of the PS as the number of basis states, denoting its size. This reduced program specification enables more efficient sampling of the quantum program distribution due to the smaller program specification rank, tackling challenge (1), and the detection of phase flip faults by projective measurements, alleviating challenge (2). We evaluate our approach on a program suite consisting of 143 quantum programs in 4 categories. Given that not all quantum programs yield final state vectors expressible in the eigenstates of mixed Hadamard bases (which we associate with uniform superpositions with real amplitudes), we select four program categories that demonstrate this to varying degrees: Grover search and graph state programs, known for their high uniformity; discrete quantum walks, characterized by their non-uniform final state vectors; and a diverse array of other quantum algorithms to investigate the impact of varying uniformity on our measurements [24, 26, 32, 68].

In our evaluation, we assess: (1) the effectiveness of our reduction approach; (2) the impact of reduced PSs on QST runtime efficiency; and (3) the impact of reduced PSs on effectiveness, as measured through mutation testing. We find that reduction is generally efficient and effective, achieving reduction rates 52.7% overall, and 27.4% in the low end for quantum walk programs and 83.7% in the high end for Grover searches. Our approach significantly improves the test runtimes compared to sampling on the computational basis, yielding a 14-fold improvement on average, from 169.9 s with computational basis sampling to 11.8 s with our approach. Particularly notable are 11-fold improvements for Grover search programs with large circuit depths and 60-fold improvements for graph state programs with large

specification ranks. In contrast, the categories of quantum walks and various quantum algorithms showed mixed results, with no improvements observed in 12.8% and 9.1% of cases, respectively. Nevertheless, quantum walks achieved a threefold improvement, reducing from 2.9 s to 1.0 s, the category of various algorithms obtained a sixfold improvement, decreasing from 1.1 s to 0.2 s. Overall for all program categories, we find that fault detection is enhanced, with mutation scores of 74.7% for various mutations compared to 54.5% using non-reduced specifications. Notably, non-reduced specifications hardly detect phase flip faults, with a mutation score of only 2.1%, while our approach achieves 36.0% overall.

To summarize, the main contributions of our paper are:

(1) A Greedy rank reduction algorithm to obtain a reduced program specification for projective measurements in a mixed Hadamard basis;
(2) An experimental evaluation of our approach using a suite of 143 quantum programs, including important application programs such as Grover searches, with qubit counts up to 16, depths up to 684, and program specification ranks up to 2^{14} computational basis states;
(3) Empirical evidence of faster sampling of the quantum program distribution through our reduced specifications; and
(4) Empirical evidence of detection of phase gate faults through projective measurements with the mixed Hadamard basis detecting previously undetectable phase flip faults.

We provide a replication package, containing the approach's source code, study subjects, experiment scripts, paper results, and appendix with additional results [47].

2 Background and Definitions

This section introduces quantum computing (QC) and provides important definitions in three parts: **Qubits and State Vectors**, **Quantum Gates** and finally **Quantum Programs and Projective Measurements**.

2.1 Qubits and State Vectors

In classical computing, bits represent information as either 0 or 1. Conversely, quantum computing applies *quantum bits* or *qubits*, which can be in a state of 0 and 1 simultaneously through *quantum superposition* [46]. Mathematically, we represent a single qubit by the two-component *state vector* $|\psi\rangle \in \mathcal{H}$, where $\mathcal{H}$ denotes the *Hilbert space* of the qubit:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \tag{1}$$

In Eq. (1), the states $|0\rangle$ and $|1\rangle$ are the *computational basis states* for the state $|\psi\rangle$, corresponding to the possible outcomes that are observed when measuring the qubit in the computational basis:

$$\{ |0\rangle, |1\rangle \} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \tag{2}$$

Coupled with the basis states are the complex numbers α_0 and α_1, known as the *probability amplitudes* of the basis states $|0\rangle$ and $|1\rangle$. We can write the probability amplitudes as the product $\alpha_m = p(m)e^{i\theta_m}$ consisting of two parameters. First, the probability $p(m)$ of observing the particular basis state $|m\rangle$ is obtained by computing the squared absolute value of the probability amplitude, i.e. $p(m) = |\alpha_m|^2$. Thus, the set of probabilities for a given quantum state defines the

quantum state's probability distribution of basis states. Second, the angle parameter $\theta_m \in \{0, 2\pi\}$ defines the complex number $e^{i\theta}$ which is the *relative phase* of $|m\rangle$.

While Eq. (1) describes single-qubit states, for n-qubit states: $q_0, q_1, \cdots, q_{n-1}$, the state vector can be written as a sum of all possible configurations of the n-qubit basis states:

$$|\psi\rangle = \sum_{j_0=0}^{1} \sum_{j_1=0}^{1} \cdots \sum_{j_{n-1}}^{1} \alpha_{j_0 j_1 \cdots j_{n-1}} |j_0 j_1 \cdots j_{n-1}\rangle . \tag{3}$$

To form n-qubit basis states, we apply the *tensor product* operation $\otimes$ [46], such that $|j_0 j_1 \cdots j_{n-1}\rangle = |j_0\rangle \otimes |j_1\rangle \otimes \cdots \otimes |j_{n-1}\rangle$.
We can write Eq. (3) more compactly as:

$$|\psi\rangle = \sum_{j=0}^{N-1} \alpha_j |j\rangle \tag{4}$$

Here, the states $|j\rangle$ represent the $N = 2^n$ basis states from Eq. (3) in compact decimal representation [46]. The relation $|j\rangle = |j_0 j_1 \cdots j_{n-1}\rangle$ maps between the decimal and binary representations of the state. For example, if $j = 2$ and $n = 3$ then $|2\rangle = |010\rangle$. Additionally, we define the probability distribution $\mathcal{P} = \{p_0, p_1, \cdots, p_{N-1}\}$ of a state vector that obeys the normalization condition:

$$\sum_{j=0}^{N-1} p_j = 1 \tag{5}$$

In Eq. (5), the probabilities of the quantum program distribution add to 1. A state vector $|\psi_e\rangle$ of a system composed of multiple qubits represents an *entangled state* if and only if it *cannot* be expressed as a tensor product of state vectors from each qubit [25, 46]. The canonical example of an entangled state is one of the *Bell states* $|\beta\rangle = 1/\sqrt{2}(|00\rangle + |11\rangle)$. On the flip side the state $1/\sqrt{2}(|00\rangle + |01\rangle)$ *can* be written as a tensor product of the states $|0\rangle$ and $1/\sqrt{2}(|0\rangle + |1\rangle)$, so it is not an entangled state. Entangled states represent correlations between qubits that are specific to quantum states and thus not possible for classical states and have been confirmed for quantum states separated by large distances [66].

Output Criterion If all basis states of a state vector are equally likely, we call Eq. (4) a *uniform* state vector. In addition, if all probability amplitudes are real numbers, we write the class of state vectors as:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{j=0}^{N-1} (-1)^{f(j)} |j\rangle \tag{6}$$

In Eq. (6), the function f determines whether the phase angle is either π or 0, resulting in a sign in front of the state of either -1 or $+1$ respectively. Because these states are simple to study and important for many applications, such as for Grover search and graph states [24, 26], we construct our reduction approach to target these kinds of states. Thus, we call Eq. (6) the *output criterion* for later reference.

2.2 Quantum Gates

As with classical computing, quantum gates transform one computational state to another. In QC, gates act as unitary matrices on the state vectors to perform computations. The *Hadamard* gate denoted by H, performs the following transformations on the computational basis states $|0\rangle$ and $|1\rangle$:

$$H|0\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle + |1\rangle\Big) = |+\rangle \qquad H|1\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle - |1\rangle\Big) = |-\rangle \tag{7}$$

In Eq. (7), the Hadamard gate initiates a uniform superposition with 50% chance of measuring either $|0\rangle$ or $|1\rangle$. The quantum equivalent of the classical *NOT* gate is named the X-gate or the bit-flip gate and is denoted by X. It acts as expected when applied to classical states, i.e., $X|0\rangle = |1\rangle$. However, the difference with acting on a quantum state can be shown by applying the X-gate to the single qubit state, as depicted in Eq. (1), which performs a bit flip on both terms in the sum. The resulting state would thus be $\alpha_1|0\rangle + \alpha_0|1\rangle$, where the probability amplitudes α_0, α_1 have exchanged places. This is an example of how QC differs from classical computing. A classical logic gate acts on a single state at a given time, whereas a quantum gate is applied to the state vector of superpositions such that both $|0\rangle$ and $|1\rangle$ are flipped simultaneously. The cost, however, is that we can only obtain either $|0\rangle$ or $|1\rangle$ with probabilities p_0 or p_1 at any given time by observing the state vector [46] through performing a measurement or a read of the state vector. We also introduce the *phase flip* gate Z, which acts on the computational basis states as $Z|0\rangle = |0\rangle$ and $Z|1\rangle = -|1\rangle$. The final gate we introduce is the rotation gate $R_y(\theta) = \exp(-i\frac{\theta}{2}Y)$, that applies a rotation of angle θ to the qubit along the y-axis on the Bloch sphere, where Y is the Pauli Y-gate [46]. Together, the Hadamard, X and R_y gate parameterize a general single qubit gate [46, 55].

2.3 Quantum Programs and Projective Measurements

A *quantum program* consists of an initial state vector $|\psi_{init}\rangle$, followed by the main program $\mathcal{U}$, which is a series of quantum gates, before performing a measurement of the final state vector $|\psi_{final}\rangle$. We have the following relationship $|\psi_{final}\rangle = \mathcal{U}|\psi_{init}\rangle$, from the evolution postulate of quantum mechanics [46, 55]. In order to transition from the final state vector to a definite state of 0s and 1s, one must perform a measurement M as the final operation of the quantum program. A measurement is a non-reversible operation and can be represented as a mapping from a state vector onto a specific basis state $|m\rangle$ from the state space, where $|m\rangle$ occurs with probability p_m. We define the basis state $|m\rangle$ as an *output* O_m of the quantum program resulting from the measurement M.

Example: Outputs of a 4-Qubit Final State Vector. Given the following 4-qubit final state vector: $|\phi\rangle = \frac{1}{\sqrt{4}}(|000\rangle + |001\rangle + |110\rangle + |111\rangle)$. We can identify four possible outputs resulting from a measurement: $|000\rangle, |001\rangle, |110\rangle$, and $|111\rangle$, where each output occurs with a probability equal to 1/4. We will use $|\phi\rangle$ as our running example to demonstrate how basis changes can affect the number of outputs after measurement.

Projective Measurements Up to this point, we've only been concerned with the basis states of the computational basis, as shown in Eq. (2). However, for this paper's approach, we also utilize projective measurements, which represent transformed measurements where we project the quantum state onto a specified subspace depending on the transformation, allowing for a reduced representation of n-qubit state vectors. We apply projective measurements that utilize *Hadamard basis* transformations (also known as the *X-basis*, as they are the eigenstates of the Pauli X gate) [46, 55]:

$$\mathbf{H} = \{|+\rangle, |-\rangle\} = \left\{\begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}}\end{pmatrix}, \begin{pmatrix}\frac{1}{\sqrt{2}}\\ \frac{-1}{\sqrt{2}}\end{pmatrix}\right\} \tag{8}$$

Equation (8) defines the Hadamard basis $\mathbf{H}$ for single-qubit states. Now, to transform into a *mixed Hadamard basis*, we rewrite the third qubit in the state $|\phi\rangle$, from our running example, in the Hadamard basis as $|\phi\rangle = 1/\sqrt{2}(|00+\rangle + |11-\rangle)$.

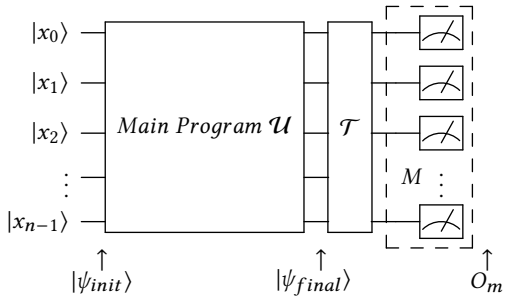

Fig. 1. Illustration of a quantum program under a projective measurement $\mathcal{T}M$.

From this point on, we will not specify which qubits are transformed but simply state that the state vector is written in the mixed Hadamard basis. The state $|\phi\rangle$ in the mixed Hadamard basis, is still mathematically equal to the state in the computational basis and, thus, results in the same outputs after a measurement. In order to perform a projective measurement in a quantum program, we must apply the transformation before we perform a normal measurement in the computational basis. Thus, we transform $|\phi\rangle$ by the basis transformation $\mathcal{T} = \mathbb{1} \otimes \mathbb{1} \otimes H$, resulting in the transformed state $|\phi\rangle_{\mathcal{T}} = 1/\sqrt{2}(|000\rangle + |111\rangle)$, by adding the appropriate Hadamard gate to the third qubit of our circuit. As with the computational basis, we can now identify the two possible outputs $|000\rangle$ and $|111\rangle$ with probabilities $1/2$ after measurement.

Generally, we define a *projective measurement with respect to a particular basis* as the basis transformation $\mathcal{T}$ on the final state vector followed by a measurement M in the computational basis:

$$\mathcal{T} = H^{x_0} \otimes H^{x_1} \otimes \cdots \otimes H^{x_{n-1}} \tag{9}$$

Equation (9) depicts the mixed Hadamard basis transformations for n-qubit state vectors considered in this paper. Here, the values x_m for $m = 0, 1, \cdots, n-1$, take the values 0 or 1 depending on the which qubits we transform with a Hadamard gate. For instance, if $n = 5$, $x_0 = 1$ means a Hadamard gate will be applied to the 1^{st} qubit, while $x_4 = 0$ means no Hadamard gate is applied to the 5^{th} qubit. Thus, exponentiating the Hadamard gate to the 0^{th} power yields the identity $H^0 = \mathbb{1}$, while the 1^{st} power gives the Hadamard gate itself $H^1 = H$. For the basis change performed to our running example state $|\phi\rangle$, we select the transformation $\mathcal{T} = H^0 \otimes H^0 \otimes H^1 = \mathbb{1} \otimes \mathbb{1} \otimes H$ from Eq. (9) and apply it to the state through adding a Hadamard gate to the third qubit just before we measure. We introduce a short-hand notation for the transformations in Eq. (9), where the character 1 replaces any identity matrix $\mathbb{1}$, and a lowercase character h replaces any Hadamard gate H. In addition, we combine the characters 1 and h without the tensor product $\otimes$, such that $H^0 \otimes H^0 \otimes H^1 \rightarrow 11h$. With this notation, we write the transformed state from the running example as $|\phi\rangle_{11h} = 1/\sqrt{2}(|000\rangle + |111\rangle)$. The notation $11h$ determines that the first and second qubits are in the computational basis, meaning 0 and 1 are read as they are, while the third qubit is in the Hadamard basis, meaning 0 and 1 are read as $+$ and $-$, respectively. In other words, the basis states of $|\phi\rangle_{11h}$ are $|00+\rangle$ and $|11-\rangle$, but they are measured as $|000\rangle$ and $|111\rangle$ in practical applications after performing the basis transformation.

Thereby, we represent the full quantum program including the projective measurement $\mathcal{M} = \mathcal{T}M$ with respect to the basis $\mathcal{T}$ as the quantum circuit from Fig. 1:

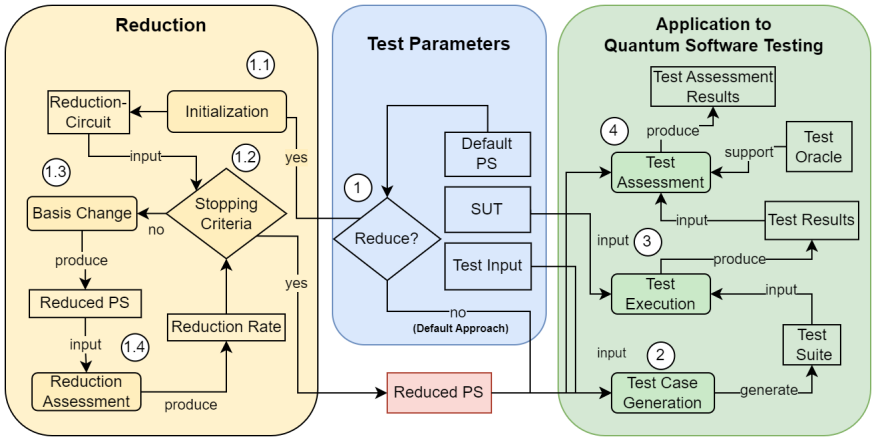

Fig. 2. Overview and Application Context of the Reduction Approach.

3 Approach

In this section, we introduce our reduction approach to QST, which consists of two components. The first component, called "Reduction", introduces a Greedy reduction algorithm designed to reduce a "Default PS" into a "Reduced PS". In the second part, called "Application To Quantum Software Testing", we describe how we utilize the "Reduced PS" in QST.

3.1 Overview

Figure 2 depicts an overview of our approach. Initially, we provide the approach's parameters in the "Test Parameters" step, i.e., a "Default PS", "Test Input", and system under test (SUT). Then, if "Reduce?" yields yes at (1), the "Reduction" component applies the reduction algorithm to the provided "Default PS". To initialize reduction in (1.1), the "Initialization" step constructs a "Reduction Circuit". Then, we pass through the "Stopping Criteria" and enter into a loop at (1.2). We iteratively perform "Basis Change"s in (1.3) to the "Reduction Circuit" to obtain a "Reduced PS". In each iteration at (1.4), we conduct a "Reduction Assessment" on the "Reduced PS", computing its rank. When the algorithm does not obtain further reductions in the rank, the stopping criteria at (1.4) terminates the loop and returns the "Reduced PS".

In the second stage, "Application to Quantum Software Testing", we provide either the "Default PS" or "Reduced PS" as input, depending on whether or not we perform "Reduction". The test process begins with the *Test Case Generation* at (2) constructing a "Test Case". Then, in the "Test Execution" step (3), the distribution of the quantum program is sampled by multiple executions, each time with a specific input state vector "Test Input" provided in the "Test Parameters" step. Finally, the "Test Assessment" at (4) evaluates these distributions, represented as "Test Results", using "Test Oracles".

3.2 Test Parameters

Here, we describe the initial step "Test Parameters" component of Fig. 2 consisting of the "Default PS", SUT, and "Test Input". These parameters provide the initial setup for our approach.

3.2.1 Program Specification

In QST, a PS plays a crucial role in validating the correctness of quantum computations by providing the expected behaviour of a quantum program [2, 37, 72, 74]. From the state vector postulate in quantum mechanics [46, 55], the state vector provides the complete description of the quantum program. Thus, we define the PS of a quantum program as its non-faulty final state vector $|PS\rangle_{\mathcal{T}}$ with respect to the basis $\mathcal{T}$ as:

$$|PS\rangle_{\mathcal{T}} = \sum_{j \in \{x|\alpha_x \neq 0\}} \alpha_j \, |j\rangle_{\mathcal{T}} \, . \tag{10}$$

In Eq. (10), $\{x|P_x \neq 0\}$ is the set of indices that correspond to the basis states $|j\rangle$ with non-zero probability amplitudes. As the PS reflects the correct final state vector of a given quantum program, we define the set of probabilities of the PS as the *Theoretical Probability Distribution*. The expression of the PS in Eq. (10) is a convenience rewrite of the general state vector form Eq. (4), such that we can define $N_{ps} \equiv |\{x|\alpha_x \neq 0\}|$ as the number of basis states in the sum of Eq. (10), which we call the *rank* of the PS. Thereby, the number of basis states in the PS may be less than the number of basis states in the state space N, i.e., $N_{ps} \leq N$.

For a non-faulty quantum program we define two types of PS: (1) the "Default PS", which is defined by the unaltered quantum program where $\mathcal{T} = \mathbb{1}$ and (2) the "Reduced PS", where the PS is reduced by a mixed Hadamard basis transformation of the type Eq. (9) such that $|PS\rangle_{\mathcal{T}} = \mathcal{T} \, |PS\rangle_{\mathbb{1}}$.

Note, when $\mathcal{T} = \mathbb{1}$ we will omit the basis notation, such that $|PS\rangle = |PS\rangle_{\mathbb{1}}$ refers to the "Default PS".

In practice, we specify a "Default PS" in the "Test Parameters" step obtained through a non-faulty version of the SUT. We provide an extensive discussion of the practical and theoretical aspects of obtaining a "Default PS" in the discussion Section 6.

3.2.2 SUT and Test Input

The SUT consists of a main program $\mathcal{U}$, (see Section 2.3), and a projective measurement with respect to the basis $\mathcal{T}$. We compare the "Default PS" to measurements of the SUT performed with respect to the computational basis, while the "Reduced PS" describes measurements on the SUT performed with respect to a mixed Hadamard basis. To run the SUT, we provide the initial state vector $|\psi_{init}\rangle$, called the "Test Input".

3.3 Reduction

In this section, we present the reduction algorithm of the "Reduction" component in Fig. 2. We define a Greedy reduction algorithm to obtain a reduced program specification, as the search space of bases in Eq. (9) grows exponentially for the number of qubits. To assess reductions, we define the rank N_{ps} of the PS as the objective function.

Running Example Throughout this section, we consider the following 3-qubit PS as a running example:

$$|\text{Default PS}\rangle = \frac{1}{\sqrt{8}}(|000\rangle + |001\rangle + |010\rangle - |011\rangle + |100\rangle + |101\rangle - |110\rangle + |111\rangle). \tag{11}$$

We chose this example state vector because it allows us to demonstrate two key properties of the basis search space that affect the runtime requirements of our algorithm design. First, it shows that a "Reduced PS" is not unique in general, so we might ask how much search space exploration is necessary for a sufficient rank reduction? Second, two different

"Reduced PS" may give the same degree of reduction, so how do we select them? For instance, given the three basis transformations $\mathbb{1} \otimes \mathbb{1} \otimes H$, $\mathbb{1} \otimes H \otimes \mathbb{1}$ and $H \otimes \mathbb{1} \otimes H$, they transform Eq. (11) into three possible reduced states:

$$|\text{Reduced PS}\rangle_{ihi} = \frac{1}{2}\Bigg(\underbrace{|0+0\rangle + |0-1\rangle + |1-0\rangle + |1-1\rangle}_{N_{ps}=4}\Bigg) \tag{12}$$

$$|\text{Reduced PS}\rangle_{iih} = \frac{1}{2}\Bigg(\underbrace{|00+\rangle + |01-\rangle + |10+\rangle - |11-\rangle}_{N_{ps}=4}\Bigg) \tag{13}$$

$$|\text{Reduced PS}\rangle_{hih} = \frac{1}{\sqrt{2}}\Bigg(\underbrace{|+0+\rangle + |-1-\rangle}_{N_{ps}=2}\Bigg) \tag{14}$$

In the two "Reduced PS" in Eqs. (12) and (13), the rank is the same, from $N_{ps} = 8$ to $N_{ps} = 4$ in the "Default PS", while in Eq. (14) it is reduced to $N_{ps} = 2$. We consider the best reduction of the three to be the latter as it achieved the smallest rank N_{ps}. Thereby, we treat the other two bases in Eqs. (12) and (13) as indistinguishable. We discuss the implications of this assumption in Section 6. In terms of runtime considerations, to obtain the better result of Eq. (14), two Hadamard gates are required, compared to only one in Eq. (12) or Eq. (13). Thus, this illustrates that a larger reduction in the rank may require more search time to find additional Hadamard gates.

In the remainder of this section, we define our Greedy reduction algorithm in Algorithm 1 with the three steps "Initialization", "Basis Change", and "Reduction Assessment", and illustrate its workings through the running example.

3.3.1 Initialization

We first create a quantum circuit, called the *reduction-circuit*, for storing the "Default PS" in the circuit and for performing basis transformations. We denote the reduction-circuit by $|RC_{\texttt{Initial}}\rangle$ and initialize it to the state vector $|PS\rangle$, representing the "Default PS". Next, on Line 2 in Algorithm 1, we initialize the basis transformation array $\mathcal{T}$ with n identity operators $\mathbb{1}$ indicating no initial transformations on the qubits. This array stores and applies transformations throughout the algorithm. Following this on Line 3, we compute the rank N_{ps} of the initial PS, storing the result in $N_{\texttt{psPrevious}}$, using the `CountBasisStates` function. We use this initial rank to determine the "Stopping Criteria". Subsequently, Lines 4 and 5 define the `search_space` and $N_{\texttt{psList}}$ arrays. The `search_space` represents the indices of the qubit register. We initialize the array $N_{\texttt{psList}}$ with n zeros, which stores the ranks of the "Reduced PS" for each transformed bases throughout the algorithm, keeping track of the reductions in each iteration.

Running Example (Initialization Step): We declare the reduction-circuit, $|AC_{\texttt{Initial}}\rangle$, that holds the state vector in Eq. (11). Now, the final state vector of the reduction-circuit is identical to the running example state vector, i.e., $|AC_{\texttt{Initial}}\rangle = |\text{Default PS}\rangle$. The remaining variables for the initialization step for our running example are specified as follows. We start by initializing the basis transformation array, $\mathcal{T} = [\mathbb{1}, \mathbb{1}, \mathbb{1}]$, to the computational basis. Next, we compute the rank of the initial PS, setting $N_{\texttt{psPrevious}} = 8$. The array of qubit indices available for Hadamard transformations is defined as `search_space` $= [0, 1, 2]$. Finally, $N_{\texttt{psList}} = [0, 0, 0]$ is used to store the ranks resulting from the transformations, maintaining the minimum rank at the end of each iteration.

Algorithm 1 Reduction Algorithm

Input: "Default PS" $|PS\rangle$, Number of Qubits n
Output: "Reduced PS" $|PS\rangle_{\mathcal{T}}$

Initialization Step:

1: $|RC_{\texttt{Initial}}\rangle \leftarrow |PS\rangle$ /* Reduction-Circuit */
2: $\mathcal{T} \leftarrow \{\ \mathbb{1} \mid \ell = 0,\ 1,\ \ldots,\ n-1\ \}$ /* Basis Transformation with Identity Operators */
3: $N_{\texttt{minPrevious}} \leftarrow \texttt{CountBasisStates}(\ |PS\rangle\ )$
4: $N_{\texttt{minCurrent}} \leftarrow N_{\texttt{minPrevious}}$
5: $\texttt{search\_space} \leftarrow \{\ \ell \mid \ell = 0,\ 1,\ \ldots,\ n-1\ \}$
6: $N_{\texttt{psList}} \leftarrow \{\ 0 \mid \ell = 0,\ 1,\ \ldots,\ n-1\ \}$ /* Set of 0s to Store Number of Basis States */

7: **while** $\neg\left(\textbf{STOPPING\_CRITERIA}\left(N_{\texttt{minCurrent}} \geq N_{\texttt{minPrevious}}\right)\right)$ **do**

8: **Basis Change Step:**

9: **for** j in search_space **do**
10: $\mathcal{T}[\texttt{j}] := H$
11: $|RC_{\texttt{Reduced}}\rangle \leftarrow \texttt{ApplyBasisTransformation}(\ |RC_{\texttt{Initial}}\rangle, \mathcal{T}\ )$
12: $N_{\texttt{psReduced}} \leftarrow \texttt{CountBasisStates}(\ |RC_{\texttt{Reduced}}\rangle\ )$
13: $N_{\texttt{psList}}[\texttt{j}] := N_{\texttt{psReduced}}$
14: $\mathcal{T}[\texttt{j}] := \mathbb{1}$
15: **end for**

16: **Reduction Assessment Step:**

17: $N_{\texttt{minCurrent}} = \texttt{min}(N_{\texttt{psList}})$ /* Find Minimum Value */
18: $N_{\texttt{minList}} \leftarrow \texttt{FindMinIndices}(N_{\texttt{psList}}, N_{\texttt{minCurrent}})$ /* Indices in $N_{\texttt{psList}}$ Equal to $N_{\texttt{min}}$ */
19: $\texttt{k} \leftarrow \texttt{RandomChoice}(N_{\texttt{minList}})$ /* Randomly Select a Minimum Index */
20: $\mathcal{T}[\texttt{k}] := H$ /* Fix the k'th Gate */
21: search_space.RemoveIndex(k)
22: $N_{\texttt{minPrevious}} \leftarrow N_{\texttt{minCurrent}}$
23: **end while**
return $\mathcal{T}$

3.3.2 Basis Change

After the initialization step follows the "Basis Change" step with a while loop on line 7, which continues until reaching the "Stopping Criteria". The stopping criteria assesses whether the previous iteration resulted in a reduction and, consequently, the algorithm terminates and returns the "Reduced PS". Inside the while loop, we initiate the basis change by iterating over the qubit indices in the search_space array. Line 10 activates a single Hadamard gate at the index j of $\mathcal{T}$, and we provide $\mathcal{T}$ and $|RC_{\texttt{Initial}}\rangle$ to ApplyBasisTransformation on line 11. ApplyBasisTransformation converts $\mathcal{T}$ into a basis transformation according to Eq. (9) and applies it to $|RC_{\texttt{Initial}}\rangle$, yielding the transformed state $|RC_{\texttt{Reduced}}\rangle$. We then compute the rank of $|RC_{\texttt{Reduced}}\rangle$ and store it at index j in $N_{\texttt{psList}}$ on line 13. Line 14 resets the Hadamard gate of the current iteration back to an identity operator at $\mathcal{T}[\texttt{j}]$. This process constitutes the core operation of the "Basis Change": we activate single Hadamard gates based on the indices from search_space, perform the basis transformation on the reduction-circuit, compute the rank of the transformed reduction-circuit, and deactivate the

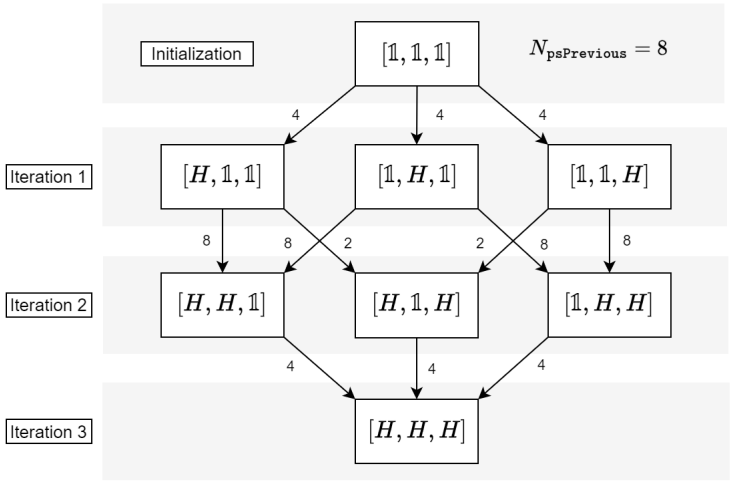

Fig. 3. Directional graph representation of the Greedy reduction Algorithm 1 for the running example in Eq. (11).

Hadamard gate again. The for-loop ends when we have performed all basis transformations specified by the indices in `search_space` and the corresponding ranks are stored in $N_{\texttt{psList}}$.

3.3.3 Reduction Assessment

After we try all the basis transformations in `search_space`, we move to the "Reduction Assessment" step. Line 17 calculates the minimum of $N_{\texttt{psList}}$, which points to the basis with the largest reduction. If there are multiple equal minimum values, we pick one randomly on lines 18 and 19. On line 20, we select an index k randomly from $N_{\texttt{minList}}$ and fix the k^{th} Hadamard gate in $\mathcal{T}$ on line 21 by removing k from `search_space`. The final stage of the "Reduction Assessment" step is to update $N_{\texttt{minPrevious}}$ by assigning it the value of $N_{\texttt{minCurrent}}$ such that the "Stopping Criteria" in the next iteration compares the next current minimum value found against the previously one.

Running Example (Basis Change & Reduction Assessment): Our example state has 8 possible basis transformations:

$$\begin{array}{llll} \mathbb{1}\otimes\mathbb{1}\otimes\mathbb{1} & \mathbb{1}\otimes\mathbb{1}\otimes H & \mathbb{1}\otimes H\otimes\mathbb{1} & \mathbb{1}\otimes H\otimes H \\ H\otimes\mathbb{1}\otimes\mathbb{1} & H\otimes\mathbb{1}\otimes H & H\otimes H\otimes\mathbb{1} & H\otimes H\otimes H. \end{array} \tag{15}$$

We perform 3 iterations of the while-loop. In iteration 1 (illustrated in Fig. 3), we performing the basis transformations: $[H,\mathbb{1},\mathbb{1}]$, $[\mathbb{1},H,\mathbb{1}]$ and $[\mathbb{1},\mathbb{1},H]$ and compute the ranks (Line 11–12 in Algorithm 1). Recall that our initial state had $N_{ps} = 8$, these three transformations all lead to a reduction in the rank by 4, which we can see from the transformed states:

$$
\begin{aligned}
|PS\rangle_{h11} &= \frac{1}{2}\left(\underbrace{|{+}00\rangle + |{+}01\rangle + |{-}10\rangle - |{-}11\rangle}_{N_{ps}=4}\right) \\
|PS\rangle_{1h1} &= \frac{1}{2}\left(\underbrace{|0+0\rangle + |0-1\rangle + |1-0\rangle + |1-1\rangle}_{N_{ps}=4}\right) \\
|PS\rangle_{11h} &= \frac{1}{2}\left(\underbrace{|00{+}\rangle + |01{-}\rangle + |10{+}\rangle - |11{-}\rangle}_{N_{ps}=4}\right)
\end{aligned} \tag{16}
$$

Thus, $N_{\texttt{psList}} = [4, 4, 4]$, which means we select one of the transformations at random on Line 20 in Algorithm 1. If we choose $[H, \mathbb{1}, \mathbb{1}]$, we remove index 0 from `search_space:=search_space=`[1,2], such that in the next iteration we only iterate over qubits 1 and 2. Then, in iteration 2 (see Fig. 3), we apply the basis transformations $[H, H, \mathbb{1}]$ and $[H, \mathbb{1}, H]$. The first choice leads to an increase in rank, back to 8 basis states, while the latter leads to the state $1/\sqrt{2}(|{+}0{+}\rangle + |{-}1{-}\rangle)$ with $N_{ps} = 2$. Thereby, we obtain $N_{\texttt{psList}} = [4, 8, 2]$ we select the second index, i.e the minimum, corresponding to the basis $[H, \mathbb{1}, H]$. Despite this being the global minimum, the algorithm continues to iteration 3 and terminates on the "Stopping Criteria" as the basis $[H, H, H]$ leads to an increase in the rank to $N_{ps} = 4$. Thus Algorithm 1 returns the "Reduced PS" $[H, \mathbb{1}, H]$. Let's explore a couple of other possibilities. If we had chosen $[\mathbb{1}, \mathbb{1}, H]$ at the end of iteration 1 instead, we also find the reduction to 2 (which we see from Fig. 3). However, if we instead choose $[\mathbb{1}, H, \mathbb{1}]$ with a reduction to the rank 4 state $1/2(|0+0\rangle + |0-1\rangle + |1-0\rangle + |1+1\rangle)$, then we are stuck at a local minimum, as the next transformations in iteration 2 are either $[H, H, \mathbb{1}]$ or $[\mathbb{1}, H, H]$, which both lead to an increase back to 8 basis states, terminating the "Stopping Criteria" and returning the local minimum basis $[\mathbb{1}, H, \mathbb{1}]$. All possible reduction paths, can be studied in Fig. 3. As we reduce the basis search space by a Hadamard gate after each iteration, our reduction algorithm finds a local or global minimum with $O(n^2)$ number of basis transformations, which we show in our appendix [47].

We then provide the "Reduced PS" as input to the "Application to Quantum Software Testing" component.

3.4 Application to Quantum Software Testing

In this section, we present the "Application to Quantum Software Testing" component in Fig. 2, which applies "Reduction" to QST and contains the following steps: "Test Case Generation" and "Test Execution".

3.4.1 Basis Dependent Test Cases

To test a quantum program, we compare its theoretical probability distribution with a sample distribution resulting from multiple executions of the program. In order to account for measurements in different bases, we define a test case as a function of the basis $\mathcal{T}$:

$$
TC(\mathcal{T}) = \left\{ |\psi_{init}\rangle, \mathcal{P}_{ps}(\mathcal{T}), Outputs \right\} \tag{17}
$$

In Eq. (17), $|\psi_{init}\rangle$ represents the input state vector, while $\mathcal{P}_{ps}(\mathcal{T})$ is the theoretical probability distribution of the quantum program (see Section 3.2.1), defined as the set of probabilities of each output basis state, but extended to include measurement with respect to the basis $\mathcal{T}$:

$$\mathcal{P}_{ps}(\mathcal{T}) = \{p_0(\mathcal{T}), p_1(\mathcal{T}), \cdots, p_{N-1}(\mathcal{T})\} \tag{18}$$

We compute the k^{th} probability in Eq. (18) by applying the basis transformation $\mathcal{T}$ to the "Default PS", then computing the squared norm of the probability amplitudes:

$$p_k(\mathcal{T}) = \left|\alpha_k(\mathcal{T})\right|^2 \tag{19}$$

Where the probability amplitudes in Eq. (19) are the amplitudes for the basis states in the "Reduced PS".

Finally, we define *Outputs* as the set of possible basis states in |Reduced PS⟩:

$$Outputs = \left\{ |j\rangle \text{ |if } j \text{ is a basis state of |Reduced PS}\rangle \right\} \tag{20}$$

Test Case and Suite Generation: We follow these steps to generate a test case:

(1) Specify an input $|\psi_{init}\rangle$ and |Default PS⟩, then obtain $\mathcal{T}$ from Algorithm 1
(2) Compute |Reduced PS⟩ = $\mathcal{T}$ |Default PS⟩
(3) Calculate the probability distribution and outputs of |Reduced PS⟩ with Eq. (19) and Eq. (20)

When steps (1) to (3) are performed, the pair $|\psi_{init}\rangle$ with $\mathcal{P}_{ps}(\mathcal{T})$ is a test case from Eq. (17).

3.4.2 Test Execution with Projective Measurements

The "Test Execution" step takes the previously generated "Test Case" and the SUT as inputs. We execute the test cases against the SUT and sample the quantum program with N_E program executions, acquiring the sample distribution Λ_{sample}, which constitutes the "Test Results" in Fig. 2. We now sample the quantum program $\mathcal{U}$, defined in Section 2.3, with projective measurements $\mathcal{M}$ in the mixed Hadamard basis $\mathcal{T}$. Thus, we define the *sample distribution* for a given test case, as a function of the basis $\mathcal{T}$:

$$\Lambda_{sample}(\mathcal{T}, N_E) = \left\{\lambda_0(\mathcal{T}, N_E), \lambda_1(\mathcal{T}, N_E), \cdots, \lambda_{N-1}(\mathcal{T}, N_E)\right\}. \tag{21}$$

In Eq. (21), the sample distribution quantifies the relative frequencies $\lambda_m(\mathcal{T}, N_E)$ of each output basis state from the quantum program measured in $\mathcal{T}$:

$$\lambda_m(\mathcal{T}, N_E) = \frac{C_m(\mathcal{T})}{N_E(\mathcal{T})}. \tag{22}$$

In Eq. (22), $C_m(\mathcal{T})$ denotes the number of occurrences of the output basis state $|m\rangle_{\mathcal{T}}$, and N_E is the total quantum program executions for a given test case. As programs with more qubits require higher sample sizes compared to programs with fewer qubits, we set the sample size N_E to be proportional to the number of basis states in the PS:

$$N_E = \gamma N_{ps}. \tag{23}$$

In Eq. (23), $\gamma \in \mathbb{R}$ determines the "Test Execution" step's sample size relative to the rank N_{ps} of the PS.

Thus, our approach provides two types of potential improvements to QST. First, by defining test cases in Section 3.4.1 as a function of the basis allows a reduction in the number of output states in the program specification which entails a reduction in the size of the test case. Second, in Eq. (23), by defining the sample size as a function of the number of basis states, a smaller test case will yield a reduction in the number of quantum program executions required for obtaining the sample distribution in Eq. (21).

Table 1. Overview table of the experimental evaluation.

Research Question	Approach Component	Experiment ID	Metric	Statistic
RQ1	Reduction	1	Reduction Rate, Runtime	Mean, Standard Deviation
RQ2	Testing	2	Speedup, Slowdown	Mean, Standard Deviation
RQ3	Testing	2	Mutation Score	Mean, Standard Deviation

Test Execution Procedure: We define the procedure for sampling the quantum program and measuring in $\mathcal{T}$ as:

(1) Prepare the quantum program $\mathcal{U}$ with the input state $|\psi_{init}\rangle$ from the test case
(2) Apply the basis transformation $\mathcal{T}$ to $\mathcal{U}|\psi_{init}\rangle$
(3) Measure in the computational basis M, resulting in the projective measurement $\mathcal{M} = M\mathcal{T}$ applied to $\mathcal{U}|\psi_{init}\rangle$
(4) An execution of the program $M\mathcal{T}\mathcal{U}|\psi_{init}\rangle$ results in an output basis state O_m
(5) Update the sample distribution $\Lambda_{\mathcal{T}}$

We repeat steps (1) to (5) according to the sample size N_{Exec}.

4 Experiment Design

This section presents the experiment design for the evaluation of our approach.

4.1 Research Questions

We pose three research questions: RQ1, RQ2, and RQ3:

RQ1: How efficiently and effectively do we obtain reduced program specifications through reduction?
RQ2: What impact does applying reduced program specifications have on the efficiency of QST?
- **a)** How efficient is QST with reduced program specifications?
- **b)** How does the reduction correlate with the QST efficiency?

RQ3: What effect does applying reduced program specifications have on the effectiveness of QST?
- **a)** How effective is QST with reduced program specifications?
- **b)** How does the reduction correlate with the QST effectiveness?

4.2 Overview of the Experiment Design

Following the nomenclature of Stol and Fitzgerald [64], we conduct two laboratory experiments, illustrated in the overview Table 1. The first (Experiment 1) applies the "Reduction" component from the approach in Fig. 2, while the second (Experiment 2) applies the "Application to Quantum Software Testing" component on a suite of 143 quantum programs. In Experiment 1, we run Algorithm 1 multiple times on the same quantum program, obtaining a distribution of the reduction rate metric and the runtime. We answer RQ1 with statistical assessments of the reduction rate distributions and the time cost of running Algorithm 1. In Experiment 2, we run the "Application to Quantum Software Testing" component of Fig. 2, where we subject the SUT to mutation testing. We conduct two testing scenarios: one with a "Reduced PS", derived from the median reduction rate resulting from Experiment 1, and another with a "Default PS". The data gathered, consisting of the metrics of testing runtime and mutation score for each scenario, are used to evaluate the efficiency posed in RQ2 and the effectiveness in RQ3, respectively.

Table 2. Overview table of the study subjects in our evaluation.

Categories	Grov	Gs	Qwalk	Var
#Programs	48	14	39	44
Qubit-Range	[6, 9]	[3, 16]	[3, 5]	[2, 8]
Depth-Range	[2, 684]	[5, 18]	[1, 154]	[3, 70]
Rank N_{ps}-Range	[32, 256]	[2, 16384]	[1, 16]	[4, 16]
Program Variations	Random Oracles	Ring Graphs	# Walks	Inputs 0-3

4.3 Random Baseline

In our evaluation, we compare our Greedy algorithm to a random search baseline, which we call Random reduction. For a given quantum program, we first evaluate using the Greedy approach, resulting in a number of executions of the reduction-circuit. Our random baseline algorithm then also performs the same number of executions, but with random samples of the exhaustive basis search space. At the end, Random selects the basis that minimizes the rank (i.e, maximizes the reduction). We then compare the results of Greedy to Random in both Experiment 1 and Experiment 2.

4.4 Study Subjects (Quantum Programs)

We select study subjects that satisfy the output criterion detailed in Eq. (6), i.e., programs where the final state vector is uniform and has real amplitudes, such that its final state vector approximates an eigenstate of a mixed Hadamard basis. Different quantum algorithms satisfy the output criterion to varying extents based on their gate composition and semantics. Graph states, consisting only Hadamard and controlled Z gates, meet this criterion often [26]. In contrast, while Quantum Fourier transforms give uniform superpositions, they yield complex amplitudes and quantum walks has the opposite problem [21, 46, 68]. Additionally, since large reductions increase runtime overhead due to Hadamard gates in the circuit for projective measurements, we aim to test our approach against programs varying in depth and number of output states (N_{ps}).

Based on these criteria, we select 145 quantum programs divided into four categories:

- Grover Search (Grov) with 48 programs, noted for their high program depth.
- Quantum Walk (Qwalk) with 39 programs, characterized by a lower satisfaction of the output criterion.
- Graph States (Gs) comprising 14 programs, marked by their high number of output states.
- Various (Var) with 44 programs, characterized by representing multiple quantum algorithms.

These categories, detailed in Table 2, source from Hein et al. [26], Johnston et al. [32], Quetschlich et al. [53]. Employing the nomenclature of Baltes and Ralph [6], our sample is theoretical rather than representative of all quantum algorithms, as we exclude algorithms like VQE, QAOA, or Hamiltonian simulation, which do not meet the output criterion. On the other hand, we *do* argue our sample is representative of programs where the output criteria are satisfied.

In our evaluation, classical simulations of quantum programs cause runtimes to scale exponentially with the number of qubits. Therefore, to maximize the depth of Grov programs and the number of outputs from Gs programs, we set upper thresholds for depth and qubit counts, as detailed in the following paragraphs. Our thresholds are the points where adding one qubit doubles the runtime and exceeds our budget. This budget sets a maximum evaluation runtime of one week, aiming to use as many or more qubits and depths than other QST evaluations in related work [28, 41, 49, 72].

Grover Search (Grov) The Grover Search program takes a set of specified output states in a search oracle, then carries out iterations of Grover operations, marking and amplifying the probability amplitudes of the specified set of output states in the oracle [32].

We establish our upper threshold by considering the empirical trial of the 9 qubit Grov programs with depths of 684 and runtime of 9 hours. To include multiple variations of these programs, going beyond this qubit count and depth becomes challenging, as runtimes double from this point for each added qubit. Thus, we use the depth of 700 and 9 qubits as the suitable upper thresholds for Grov. With respect to these thresholds, we generate programs incrementally from 6 to 9 qubits and generate 360 program variations per qubit with random oracles from 256 possible oracles. For each variation, we randomly select sets and perform Grover operations until non-input state amplitudes are below 10^{-4}. In order to achieve this precision, we allow a non-optimal number of Grover iterations. This allows our sample to simulate large depths, where many iterations are needed for the optimal case. Excluding two variations for exceeding depth 700, we obtain 46 study subjects from the *Grov* category.

Graph States (Gs) These programs represent a quantum state encoding of a graph $G = (V, E)$, with vertices V and edges E [3, 26]. This process yields low-depth Gs programs with a consistent 2^{n-2} output states, making them ideal as large rank programs. In our selection, we focus on ring graphs for their edge set $E = \{(0, 1), (1, 2), \cdots, (n-1, 0)\}$, given a qubit count n.

As with Grov, we use the empirical trial for the 16 qubit Gs program with runtime of $4937s$. Thus, from the qubit threshold of 16, we generate programs incrementally from 3 to 16 qubits.

Quantum Walk (Qwalk) We aim to include programs in our evaluation that do not necessarily meet the output criteria specified in Eq. (6). Discrete quantum walks, the quantum equivalents of classical random walks, are particularly suitable for this purpose as they typically do not converge to uniform superpositions [21, 68].

Unlike the Grov and Gs programs, maximization of qubit counts and depths is not required for Qwalk programs. Our goal is to evaluate our approach using programs that may not satisfy our predefined output criteria. Consequently, we have chosen to include Qwalk programs with qubit counts ranging from 5 to 8, generating 12 variations per qubit count and varying the number of walks from 1 to 15 to ensure a diverse sample of program variations.

Various (Var) In the Var category, we include programs from multiple quantum algorithms to evaluate our approach. We analyze final state vectors using the QCengine API [32] to select these programs. Initially, we consider 52 programs, but exclude simple demonstrations (e.g., Examples: 2-1, 2-2, 2-3, 3-1), programs with non-reducible single outputs (e.g., Examples: 3-4, 5-6, 10-2, 12-2, 12-4, 14-GT, 14-BV, 14-S), and any exceeding 16 qubits (e.g., Examples: 4-2, 11-2, 11-4, 11-6, 12-1). We detail the selection in Table 3. For each program, we generate four manually verified valid input states from 0 to 3 to ensure adequate sample diversity. We refer to Honarvar et al. [28] for more details.

4.5 Experiment Setup

This section details the experimental setup for evaluating RQ1–RQ3, according to the overview in Table 1.

4.5.1 Experiment 1 for RQ1 Evaluation

In Experiment 1, we conduct r_1 repetitions of the reduction algorithm (Algorithm 1) on each study subject. We first perform Experiment 1 for the Greedy approach and store the number of objective function calls. Then, we perform Experiment 1 or the Random baseline, giving the number of objective function calls from the Greedy run as input.

Table 3. Overview of the Var category study subjects used for our evaluation, sourced from [54].

Example id	Program	#Qubits	Depth	N_{ps}
3-3	Phase kickback	3	3	4
3-5	Custom conditional-phase	2	6	4
4-1	Basic teleportation	3	4	8
5-2	Adding two quantum integers	6	8	4
5-3	Add-squared	6	14	4
5-4	Quantum conditional execution	6	11	4
5-5	Quantum conditional phase flip	5	11	8
6-3	Multiple flipped entries	4	69	4
10-1	Phase Logic 1	4	11	8
10-4	Unsatisfiable 3-Sat	7	41	8
11-3	Drawing into small tiles	8	17	16
12-3	Shor step-by-step	8	13	16

Metric RQ1 – Reduction Rate: At the end of each repetition of Experiment 1, we calculate the *Reduction Rate R* defined as the ratio between the ranks $\tilde{N}_{ps}$ and N_{ps} of the "Reduced PS" and "Default PS" respectively:

$$R = 100(1 - \frac{\tilde{N}_{ps}}{N_{ps}}). \tag{24}$$

While Arcuri and Briand [5] suggest 30 repetitions as a rule of thumb, our approach's efficiency allows us to exceed this by performing $r_1 = 100$ repetitions of Experiment 1 for each program. This is feasible because our approach requires at most $O(n^2)$ objective function calls. After conducting r_1 repetitions for each of the 143 study subjects across our four program categories, we collect four sets of *Reduction Rate Distributions* to evaluate our approach's effectiveness.

Statistical Analyses (Experiment 1): We apply the Mann-Whitney U test to statistically analyze the significant difference between the Greedy approach and Random baselines. The null hypothesis H_0 states that the Greedy and Random distributions are the same, while the alternative hypothesis H_1 states that the Greedy distribution is stochastically different from the Random distribution. To assess the strength of statistical significance, we apply the Vargha-Delanay effect size $\hat{A}_{12}$ [5, 67]. If $\hat{A_{12}} = 0.5$, there is no difference between Greedy and Random, while $\hat{A}_{12} > 0.5$ favors the Greedy approach. For a given effect size, we define four nominal magnitude categories based on the scaled value $\hat{A}_{12}^{scaled} = 2(\hat{A}_{12} - 1/2)$ [27, 36]. The nominal magnitude of the effect sizes are:

(1) *Negligible* (N) for $|\hat{A}_{12}^{scaled}| < 0.147$
(2) *Small* (S) for $0.147 \leq |\hat{A}_{12}^{scaled}| \leq 0.33$
(3) *Medium* (M) for $0.33 \leq |\hat{A}_{12}^{scaled}| < 0.474$
(4) *Large* (L) for $|\hat{A}_{12}^{scaled}| \geq 0.474$

We consider results as statistically significant if p-value ≤ 0.05 and the effect size $\hat{A}_{12}$ is greater than magnitude (N).

4.5.2 Experiment 2 for RQ2–RQ3 Evaluation

We run Experiments 1 and 2 with Qiskit version 0.45.1.

As current benchmark suites for quantum programs are in early phases, containing limited faulty programs [77]. In Experiment 2, we apply mutation testing to obtain a faulty program suite using three distinct single qubit mutation operators: a bit flip (X-gate), a phase flip (Z-gate), and a probability perturbation along the y-axis (R_y-gate). These

operators enable us to simulate all possible single qubit faults by covering the Bloch sphere [46]. We manually generate mutations because, to our knowledge, no existing mutation tools support the projective measurements required by our program specifications [19, 41]. We insert these mutation operators between the main quantum program $\mathcal{U}$ (see Fig. 1) and the projective measurement $\mathcal{M}$ to simulate single gate faults at the end of the circuit. To ensure a comprehensive representation of gate insertions across the 16 potential locations (considering the maximum qubit count of 16 in our study subjects), we create 90 mutants at random locations for each mutation operator. For R_y, we also need to include a representative sample of the angle θ. We achieve this by 30 randomly assigned values of the angle θ. Consequently, we generate 2145 mutants (15 mutants for each of the 143 programs). For each mutant, we perform r_2 repetitions of the "Application to Quantum Software Testing" component in Fig. 2 for each type of PS, default and reduced. For the "Reduced PS", we apply our basis dependent test case generation and execution procedures in Section 3.4.1 and Section 3.4.2. Thereby, obtaining sample distributions, ready for the "Test Assessment" step. Following the best practice by Arcuri and Briand [5], we perform $r_2 = 30$ repetitions of Experiment 2 for all mutants, obtaining 30×2145 test results for the three cases of "Default PS" and "Reduced PS" with both Greedy and random baseline. Furthermore, we aim for a minimal sample size rule that is comparable across program specification sizes. Inspired by the 10 times rule [31], we set $\gamma = 10$ from Eq. (23), performing 10-fold samples of the size of the PS.

We perform a single circuit execution for each sample, as a real quantum computer. Some quantum SDKs, such as IBM's Qiskit offer a way to perform multiple samples from a given circuit execution.

Test Assessment: We apply the following two test oracles from related work [28, 41, 49, 71–73], incorporating projective measurements:

Wrong Output Oracle (WOO). The first test oracle, f_{woo}, validates whether an output basis state $|m\rangle$ obtained from a test execution is a basis state of the PS:

$$f_{woo}\Big[\,|m\rangle\,, TC(\mathcal{T})\Big] = \begin{cases} 1 & \text{if } |m\rangle \notin Outputs \\ 0 & \text{else} \end{cases} \tag{25}$$

In Eq. (25), f_{woo} takes the observed output state $|m\rangle$ and a test case (see Eq. (17)) as input and fails ($f_{woo} = 1$) if the output cannot be found among the output states of the PS.

Probability Distribution Oracle (PDO). The second oracle, PDO, conducts a chi-square hypothesis test to compare the sample distribution Λ_{sample} against the theoretical distribution $\mathcal{P}_{ps}$ from the PS [50]:

$$f_{pdo}\Big[\mathcal{P}_{ps}(\mathcal{T}), \Lambda_{sample}(\mathcal{T}, N_E)\Big] = \begin{cases} 1 & \text{if } p_{value} < \alpha \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

In Eq. (26), the null hypothesis posits no significant difference between the sample and theoretical distributions, while the alternative hypothesis suggests that such differences are present according to the significance level $\alpha = 0.05$.

Now, after applying the two test oracles (Eq. (25) and Eq. (26)), we obtain the "Test Assessment Result".

Metric RQ2 – Runtime Speedup & Slowdown: We define the *Test Runtime*, as the time taken to run the Greedy approach, "Test Execution" and "Test Assessment" (either WOO or PDO oracle fails) steps. For the "Default PS" case,

we only run the latter two steps as we perform no reduction. Thus, we define the *Default Test Runtime* T_{Def} and the *Reduction Test Runtime* T_{Red} from which we define our runtime metrics *Speedup* and *Slowdown* for evaluating RQ2:

$$Speedup = \frac{T_{Def}}{T_{Red}} \qquad Slowdown = -\frac{T_{Red}}{T_{Def}} \tag{27}$$

In Eq. (27), we define the *Speedup* (%) as the ratio between the default and reduction test runtime. Similarly, we define the *Slowdown* (%) as the ratio between the reduction and default test runtime: If $T_{Def} > T_{Red}$, we obtain a Speedup in test runtime from using the "Reduced PS"; otherwise, the "Reduced PS" results in a Slowdown of test runtime.

Metric RQ3 – Mutation Score: To evaluate RQ3, we define the *Mutation Score* [13, 19, 41] as the percentage of killed mutants for a given quantum program.

$$MUT = 100\frac{\#Killed\ Mutants}{15} \tag{28}$$

In Eq. (28), we divide the number of killed mutants by 15, as we have 15 mutants per quantum program.

Statistical Analyses (Experiment 2): In addition to the statistical analyses, p-value, and effect size defined in Section 4.5.1, we employ the Kruskal-Wallis test [35] for comparing more than two approaches. When Kruskal-Wallis finds significant differences between the distributions, we employ pairwise Mann-Whitney U (MWU) tests along with effect size measures to assess which distributions differ. We compute the Spearman rank coefficient to determine correlation between reduction rate and runtime efficiency or mutation score effectiveness, whose value is between -1 and 1, where -1 indicates no monotonic relationship and 1 indicates the strongest relationship. We adopt the following magnitude categories for correlation in order to draw categorical conclusions about our data [58]:

(1) *Negligible* if $r_s \in [0.00, 0.10]$
(2) *Weak* if $r_s \in [0.10, 0.39]$
(3) *Moderate* if $r_s \in [0.40, 0.69]$
(4) *Strong* if $r_s \in [0.70, 0.89]$
(5) *Very Strong* if $r_s \in [0.90, 1.0]$

4.6 Threats to Validity

We structure the threats to validity along the types: construct, internal, and external [63].

Construct Validity: The biggest threat to the construct validity is whether our metrics represent efficiency and effectiveness. While runtime is a straightforward measure of efficiency, it might not capture all dimensions of testing effectiveness. Such as the time taken to set up a test environment. This includes obtaining a "Default PS" for the correct program by experimental or mathematical methods [11, 78]. This may be alleviated by constructing efficient software tools and processes for ease of application. We discuss the challenge of obtaining a PS in a later section. In addition, although mutation score, provides a direct measure of the test case's ability to identify errors, our mutation operator set of the X gate, Z gate and R_y may not represent all types of faults. Although the set forms a universal single qubit gate, we may not detect certain two-qubit gate faults with our reduction approach, which are essential for entanglement in quantum computation [46].

Internal Validity: When it comes to internal validity, a crucial threat comes from potential new faults being introduced from the reduction basis being added onto the end of the circuit in order to perform the projective measurement. This threat may cause systematic faults that are introduced by the reduction approach and not by controlled experimental design, leading to a falsely high mutation score for the reduction approach. This can occur by the oracles becoming over-sensitive. We mitigated this threat through calibrating the experiment by performing a fault free run of the Default and the reduction approach and observing negligible failure rates. In addition, the 10 times rule which inspired our choice of sample size, has been found to give inadequate sample estimates [22]. While our calibration run helps mitigate this threat we still may fail to detection subtle distribution perturbations caused by R_y faults through inadequate sample sizes.

External Validity: Our approach is limited to specific program categories in addition to our choices for qubit and depth ranges. In addition, although we conduct our evaluation using Qiskit, the generalizability of our results to other quantum software development kits like CirQ or Forest is not significantly threatened. This is because our approach primarily requires gate operations and computational basis measurements, which facilitate the projective measurements essential to our methodology. However, these SDKs must be gate-based to ensure the generality of our results.

As real quantum devices in the current NISQ era contain noise, a crucial threat to the external validity is our usage of an ideal simulator. Thus, if noise were present in our experiment, the presence of noise will be treated as faults of the quantum programs. To effectively eliminate this threat, we see two mitigation methods. The first is by modifying the oracles to accept noise tolerance thresholds as inputs, then using a known noise model representing the backend to be tested. Or by using a single PDO type oracle modified to accept all possible output states, effectively eliminating the WOO oracle. The second is to apply noise mitigation techniques [8], and in the future error correction [60]. For our experiment, we assume an ideal simulator, while the former cases we discuss further in our future work. While our results are specific to the program categories in our experiment, our evaluated program categories represent real-world quantum programs, such as Grover searches, projected to be a crucial quantum algorithm for the quadratic speedup of unstructured search [17, 34, 40, 75, 77]. However, a common challenge in Grover search is the construction of the search oracle which is domain specific [59, 62]. While our results hold for the specific search oracle construction applied in our experiment, we cannot claim that our results generalize to all types of search oracles. Given that Grover search is a type of quantum walk on a bipartite graph [56], we suggest our reduction approach can be applied to certain types of discrete quantum walks that satisfy our output criterion, such as applied in hypercube quantum search [51], where the output states are uniform superpositions.

Next to consider is the generality of our results to programs with large specification sizes, such as our Gs programs. Graph states, including cluster and complete graphs, often meet our output constraints effectively by design [23, 33, 38]. Therefore, our evaluation of ring graphs provides a strong basis for extending our results to both complete and cluster graphs.

5 Results and Analyses

In this section, we present the experimental results for each research question.

5.1 RQ1: Reduction Efficiency and Effectiveness

In the **Greedy** and **Random** columns in Table 4, we observe that the Greedy approach achieved 52.7% average reduction rate, while random achieved 46.7%. The Gs category exhibits the largest reductions at 83.7% for Greedy and 75.1%

Table 4. The gray columns **Greedy [%]** and **Random [%]** show the overall average reduction rates and standard deviations. The **Different (X) [%]** columns show the success rates of MWU tests and effect size categories (S, M or L) defined in Section 4.5.1, depicting the percentages where we find Greedy and Random are significantly different, in the format of x/y, where x is the percentage where Greedy outperformed Random, and y is the reverse. **Different (NN) [%]** shows the sum of the success rates for significant effect size categories (S), (M) or (L).

Category	#Qubits	Depth	Greedy [%]	Random [%]	Different (NN) [%]	Different (S) [%]	Different (M) [%]	Different (L) [%]	Equal [%]
All	[2, 15]	[3, 684]	52.7 ± 28.2	46.7 ± 28.5	62.2/2.8	16.8/2.1	9.1/0.7	36.4/0.0	35.0
Grov	[6, 9]	[30, 684]	66.8 ± 8.4	62.4 ± 8.1	82.6/0.0	8.7/0.0	13.0/0.0	60.9/0.0	17.4
Qwalk	[3, 5]	[14, 154]	27.4 ± 23.1	22.5 ± 22.8	35.9/5.1	12.8/2.6	7.7/2.6	15.4/0.0	59.0
Var	[2, 8]	[3, 70]	49.8 ± 29.8	42.0 ± 30.2	54.5/4.5	27.3/4.5	6.8/0.0	20.5/0.0	40.9
Gs	[3, 15]	[5, 17]	83.7 ± 16.9	75.1 ± 22.3	92.9/0.0	21.4/0.0	7.1/0.0	64.3/0.0	7.1

Table 5. Summary of runtime results. The gray columns **Greedy [ms]** and **Random [ms]** show the average reduction runtime and standard deviations. The remaining columns to the right are described as in Table 4.

Category	#Qubits	Depth	Greedy [ms]	Random [ms]	Different (NN) [%]	Different (S) [%]	Different (M) [%]	Different (L) [%]	Equal [%]
All	[2, 15]	[3, 684]	732.2 ± 4013.7	743.9 ± 4083.0	18.2/68.5	6.3/9.8	4.2/14.7	7.7/44.1	13.3
Grov	[6, 9]	[30, 684]	484.8 ± 600.2	471.2 ± 587.3	19.6/63.0	8.7/15.2	4.3/26.1	6.5/21.7	17.4
Qwalk	[3, 5]	[14, 154]	70.8 ± 30.2	71.9 ± 30.0	7.7/87.2	2.6/10.3	5.1/5.1	0.0/71.8	5.1
Var	[2, 8]	[3, 70]	53.8 ± 32.3	56.3 ± 33.7	29.5/54.5	6.8/0.0	4.5/9.1	18.2/45.5	15.9
gs	[3, 15]	[5, 17]	5555.3 ± 11813.5	5711.7 ± 11995.4	7.1/78.6	7.1/21.4	0.0/21.4	0.0/35.7	14.3

for Random. The worst performing category is the Qwalk with 27.4% for Greedy and 22.5% for Random. Reduction rates vary widely across program categories as indicated by the large standard deviations. This can be explained due to the wide qubit ranges. For example, the average runtime for the 16 qubit programs of Gs is 44,088s while only 65s for the 6 qubit programs. More detailed average results by qubit count are found in the appendix [47]. The Greedy method typically outperforms Random, with the largest gap at 8.6 (pp) in the Gs category between the approaches, favoring Greedy. We find the smallest gap in the Grov category, showing more similar performances. The lowest average reduction rates are in the Qwalk category, with 27.4% for Greedy and 22.5% for Random.

We compare the Greedy and Random approaches using the statistical tests shown in the last five columns. For the **Different (NN) [%]** column in Table 4, we find that for all categories, Greedy is better than Random in 62.2% of cases, while Random is only better than Greedy 2.8% of the time. By category, we find that Greedy outperformed Random with a large margin for Grov and Gs in 82.6% and 92.9% of cases, with Random never being better. While for Qwalk and Var, Greedy is better for 54.5% and 35.9% of cases with Random being better for only 5.1% and 4.5%. Moving into the **Different (X) [%]** columns where (X) is the effect size magnitude category. Here, we find that more than 60% of the magnitude categories for Grov and Gs are of the type (L). For Qwalk and Var, however, the magnitudes are more evenly spread out between (S) and (L), with slight lower occurrence of (M). The last column, **Equal [%]**, indicates the occurrence of tests where Greedy and Random performed the same. Here, we find that for all categories, Greedy and Random are equal in 35.0% of cases. By category, we observe, in accordance with the results in the **Different (X)** columns, that Greedy and Random performed equally in 7.1% of cases for Grov and 17.4% of cases for Gs. In addition, we find equal performance for a clear majority of the cases at 59% for Qwalk and 40.9% for Var.

The runtime comparison in Table 5 indicates that Greedy and Random methods have similar average runtimes across all programs, from the **Greedy [ms]** and **Random [ms]** columns, displaying runtime in milliseconds. This is expected, as described in Section 4.5.1, we design Experiment 1 such that Greedy and Random perform the same number of objective function calls. However, the statistical tests reveal that Greedy still shows lower average runtimes overall in the **Different (NN) [%]** column, signifying Greedy slightly outperforming Random in 68.5% of the cases, while Random

Table 6. Summary of testing runtime results for all program categories. To the right of the gray columns, the headers **Default–Greedy**, **Default–Random** and **Greedy–Random** show the results of the statistical tests in the **Different (X)** and **Equal** columns for the respective pair of approaches. For each approach comparison, we show the success rate of pairwise statistical tests in the same column format as for RQ1 with **Different** and **Equal** columns, but only including the (NN) effect sizes.

						Default–Greedy		Default–Random		Greedy–Random	
Category	**#Qubits**	**Depth**	**Default [s]**	**Greedy [s]**	**Random [s]**	**Different (NN) [%]**	**Equal [%]**	**Different (NN) [%]**	**Equal [%]**	**Different (NN) [%]**	**Equal [%]**
All	[2, 15]	[3, 684]	169.9 ± 473.7	11.8 ± 61.2	15.6 ± 73.8	93.7/0.0	6.3	93.0/0.0	7.0	3.5/0.0	96.5
Grov	[6, 9]	[30, 684]	383.1 ± 503.3	33.4 ± 104.2	36.6 ± 113.5	100.0/0.0	0.0	100.0/0.0	0.0	2.2/0.0	97.8
Qwalk	[3, 5]	[14, 154]	2.9 ± 4.2	1.0 ± 2.0	1.0 ± 2.1	87.2/0.0	12.8	89.7/0.0	10.3	5.1/0.0	94.9
Var	[2, 8]	[3, 70]	1.1 ± 2.1	0.2 ± 0.3	0.3 ± 0.5	90.9/0.0	9.1	86.4/0.0	13.6	0.0/0.0	100.0
Gs	[3, 15]	[5, 17]	464.8 ± 1027.7	7.7 ± 16.7	35.1 ± 100.4	100.0/0.0	0.0	100.0/0.0	0.0	14.3/0.0	85.7

being better in 18.2% of the cases. We find these results to be mostly consistent also by category and evenly spread out across magnitude categories.

RQ1 Summary: We find the reduction approach to be generally efficient and effective, with the Greedy approach outperforming the Random one in average reduction rate, runtime and consistency. However, the results vary by category: Grov and Gs categories achieve the highest and most consistent reduction rates, while Qwalk and Var exhibit significantly lower performance. Runtimes across categories favor the Greedy over the Random approach.

5.2 RQ2: Impact of Reduction on QST Efficiency

Here, we present the results for RQ2a and RQ2b.

5.2.1 RQ2a: Test Efficiency

In Table 6, we present the average test runtimes in seconds by program category in the gray columns **Default**, **Greedy** and **Random**.

Test Runtimes. Overall, we find that the Default approach, obtained an average test runtime of 169.9s. Then, after we apply the reduction, we achieve an average test runtime of 11.8s for Greedy and 15.6s for random. By category, we find the same patterns. The Default approach generally requires a considerably higher runtime to achieve a test result compared to the reduction approaches.

Default–Greedy Comparison: We consider the statistical tests, comparing Default to Greedy. In the **Different (nn)** column, we observe that the Default test runtimes are lower in 93.7% of the cases overall. By category, we find that Default is slower for 100% of the cases for Grov and Gs, while 90.9% and 87.2% for Var and Qwalk. Conversely, we find no occurrences where Greedy is slower than Default. From the **Equal** column, we find that the approaches are equally fast in 6.3% of cases overall. by category, Greedy is always faster for Grov and Gs and equal to Random for 12.8% and 9.1% of cases for Qwalk and Var.

Default–Random Comparison: For the comparison between Default and Random, we find very similar results to the comparison of Default with Greedy.

Greedy–Random Comparison: In comparing Greedy to Random directly in the **Different (nn)** column, we find that Greedy is for the most part similar to Random. However, while Greedy is faster than Random only 3.5% overall, by category we find that that Greedy is faster for 14.3% of cases for Gs.

Table 7. Spearman correlation r_s between reduction rate and runtime improvement over Default.

			Correlation r_s		p-value		Correlation Magnitude	
Category	**#Qubits**	**Depth**	**Greedy**	**Random**	**Greedy**	**Random**	**Greedy**	**Random**
All	[2, 15]	[3, 684]	0.43442	0.46043	0.0e+00	0.0e+00	Moderate	Moderate
Grov	[6, 9]	[30, 684]	0.08752	0.06189	1.7e-36	5.0e-19	Negligible	Negligible
Qwalk	[3, 5]	[14, 154]	0.29152	0.29155	0.0e+00	0.0e+00	Weak	Weak
Var	[2, 8]	[3, 70]	0.3276	0.3714	0.0e+00	0.0e+00	Weak	Weak
Gs	[3, 15]	[5, 17]	0.77997	0.60156	0.0e+00	0.0e+00	Strong	Moderate

Relative Runtime improvement over Default: Similar to RQ1, the relatively large standard deviations stem from the fact that the average overall runtime includes runs from various qubit counts and depths, resulting in large test time differences. To account for differences in qubit count and depth, we compute the **Runtime Improvement over Default**, (denoting the metrics Speedup or Slowdown defined in Eq. (27)), between the Default test runtime and the test runtime of the reduction approach Greedy or Random. We compute these ratios for each test result and depict the results as the boxplots in Figs. 4a to 4d by category. First, from visual inspection, we observe a clear divide in improvement between the Grov and Gs categories and the Qwalk and Var categories. For the latter categories, we observe that all boxes have their lower whiskers reaching into the negative lower half of the y-axis, indicating that the reduction approaches performed worse than the Default for a significant number of cases. Still, however, the boxes stay above the y-axis, indicating that most (72.5% for Qwalk and 81.6% for Var) experienced an increase in runtime due to reduction while the remaining 27.5% for Qwalk and 18.4% for Var experienced a slowdown. On the other hand, we find only 2.8% and 6.9% slowdowns Grov and Gs. Furthermore, the runtime improvement over Default for Grov reach median values of just below 400 for Greedy, followed by Random at closer to 300. Given the same qubit range in Gs, i.e from 6 to 9 qubits, the maximum median improvement over Default is around 50 for both Greedy and Random.

For the reductions of Gs, we can make two observations regarding the differences between the Greedy and Random approaches in Fig. 4b. First, the improvements over Default of the reduction approaches seem to plateau at below 100. Second, the runtime improvements for Greedy become more consistently higher for higher qubit counts. We see this from the first quartile of Greedy increasing consistently for higher qubit counts, while the first quartile for Random stays constant.

5.2.2 RQ2b: Correlation Between Reduction and Efficiency

Next, we investigate the relationship between reduction and testing efficiency. Thus, in Table 7, we display the Spearman correlations r_s between the reduction rate and the runtime improvement over Default along with the correlation magnitude categories from Section 4.5.1.

Overall, we find a Moderate correlation of 0.43 for Greedy and 0.46 for Random. This indicates that overall, a significant relationship exists between a reduction rate and obtaining an improved test runtime over Default. However, by category, we notice that Grov, which achieved the fastest runtimes after reduction, has Negligible correlation. Conversely, Gs achieved the strongest correlation. Thus, given the very high depths of Grov programs compared to Gs, this means that high depth paired with reduction, could also be important to achieve runtime improvements. We also note for Gs that we find a Strong correlation when applying the Greedy approach, followed by Random with a

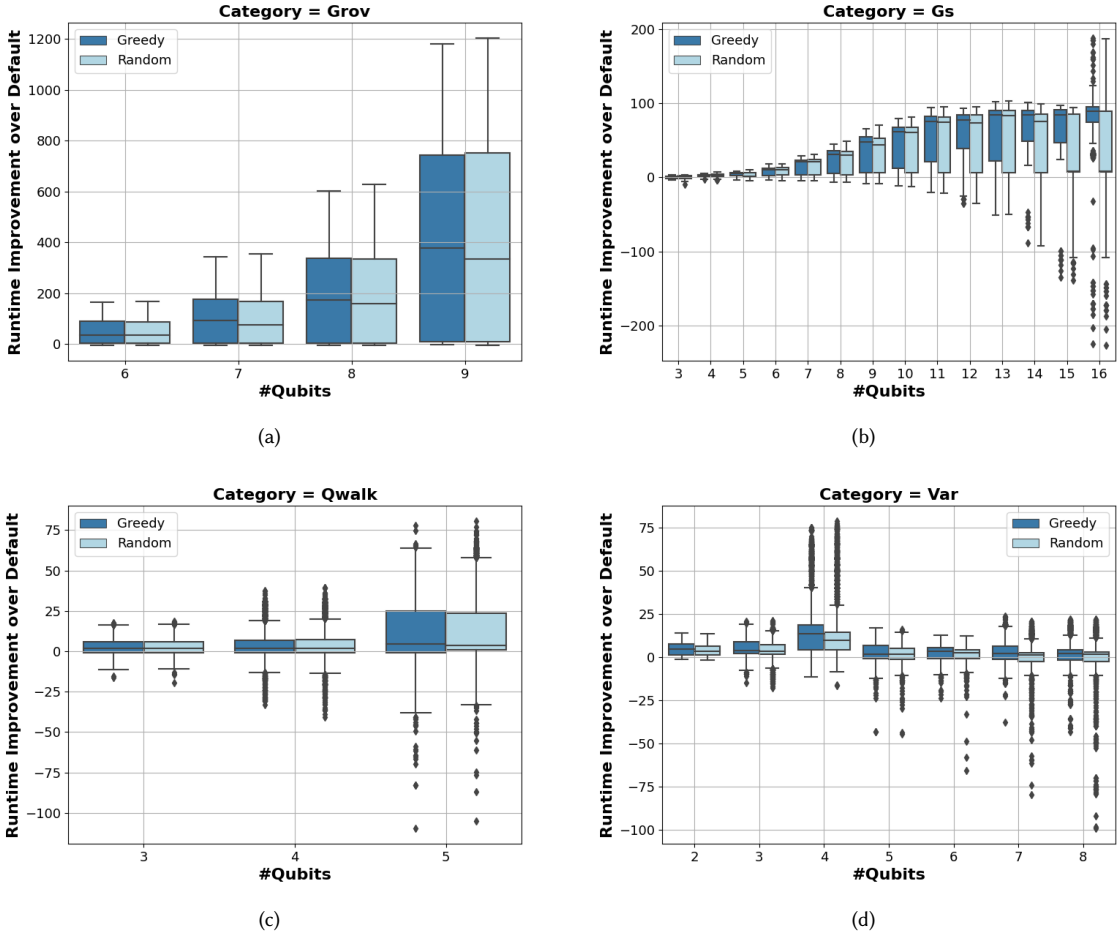

Fig. 4. Boxplots of runtime improvement over Default when we apply reduction to QST by the number of qubits. Speedups are above the x-axis and Slowdowns are below.

Moderate correlation. The categories Qwalk and Var exhibit Weak correlations. We see from the **p-value** column, that all correlations are statistically significant with p-values $\sim 10^{-19}$.

RQ2 Summary: Our results show that our reduction approach improves QST efficiency from 169.9s with the Default approach to 11.8s with reduction, but the impact varies by program characteristics. Both the Greedy and Random approaches generally improve on the Default approach, with Greedy favored in Gs programs for large specification sizes. We also observe the largest improvements for Grov programs, with high circuit depth. While Qwalk, and Var also experienced improvements to efficiency, we find no improvements to these categories in 27.5% and 18.4% of cases respectively, compared to 2.8% for Grov and 6.9% for Gs. The correlation between reduction and runtime improvements is Moderate overall.

Table 8. We show the average mutation score and standard deviations in the gray columns **Default [%]**, **Greedy [%]** and **Random [%]**. In the **DGR** column, we show the p-value result from a Kruskal-Wallis test between the Default, Greedy and Random approaches. To the right, we depict the results of the pairwise statistical tests between the respective approaches.

							DGR	Default-Greedy			Default-Random			Greedy-Random		
Category	**Mutant Type**	**#Qubits**	**Depth**	**Default [%]**	**Greedy [%]**	**Random [%]**	**p-value**	**p-value**	**Â$_{12}$**	**Magnitude**	**p-value**	**Â$_{12}$**	**Magnitude**	**p-value**	**Â$_{12}$**	**Magnitude**
All	all	[2, 15]	[3, 684]	54.5 ± 22.5	74.7 ± 9.7	74.0 ± 9.9	0.0e+00	0.0e+00	0.206	(L)	0.0e+00	0.219	(L)	0.00129	0.52	(N)
All	X	[2, 15]	[3, 684]	66.7 ± 42.1	65.5 ± 32.5	64.8 ± 32.9	1.8e-15	3.2e-12	0.541	(N)	1.7e-13	0.543	(N)	0.3887	0.505	(N)
All	Z	[2, 15]	[3, 684]	2.1 ± 9.0	36.0 ± 32.4	33.1 ± 31.6	0.0e+00	0.0e+00	0.192	(L)	0.0e+00	0.21	(L)	5.9e-05	0.524	(N)
All	R_y	[2, 15]	[3, 684]	67.9 ± 29.1	90.7 ± 10.8	90.7 ± 10.8	0.0e+00	0.0e+00	0.255	(L)	0.0e+00	0.256	(L)	0.90403	0.501	(N)
Grov	all	[6, 9]	[30, 684]	72.5 ± 10.5	77.4 ± 9.6	77.3 ± 9.7	1.3e-35	1.8e-28	0.381	(S)	8.6e-28	0.383	(S)	0.8789	0.502	(N)
Grov	X	[6, 9]	[30, 684]	96.9 ± 15.8	58.7 ± 30.1	62.0 ± 29.5	0.0e+00	3.0e-302	0.861	(L)	1.6e-278	0.842	(L)	0.00657	0.472	(N)
Grov	Z	[6, 9]	[30, 684]	3.9 ± 12.9	43.8 ± 30.1	39.3 ± 29.5	0.0e+00	1.9e-294	0.136	(L)	1.2e-262	0.161	(L)	7.3e-05	0.541	(N)
Grov	R_y	[6, 9]	[30, 684]	87.3 ± 15.1	94.8 ± 8.0	95.1 ± 7.8	1.8e-61	3.5e-43	0.363	(S)	4.5e-48	0.355	(S)	0.30569	0.491	(N)
Qwalk	all	[3, 5]	[14, 154]	51.5 ± 16.1	73.1 ± 8.9	73.1 ± 9.1	0.0e+00	8.3e-236	0.112	(L)	2.5e-234	0.114	(L)	0.98696	0.5	(N)
Qwalk	X	[3, 5]	[14, 154]	84.0 ± 31.2	75.9 ± 30.0	74.9 ± 31.1	1.9e-26	9.5e-22	0.597	(S)	1.4e-22	0.599	(S)	0.61458	0.505	(N)
Qwalk	Z	[3, 5]	[14, 154]	1.4 ± 6.6	23.3 ± 28.0	24.8 ± 30.0	5.0e-148	9.8e-132	0.276	(M)	3.0e-132	0.275	(M)	0.43133	0.491	(N)
Qwalk	R_y	[3, 5]	[14, 154]	57.4 ± 22.8	88.8 ± 11.1	88.7 ± 11.1	0.0e+00	5.2e-242	0.109	(L)	9.8e-241	0.11	(L)	0.62748	0.505	(N)
Var	all	[2, 8]	[3, 70]	38.1 ± 25.0	72.6 ± 9.6	71.6 ± 9.9	0.0e+00	1.7e-297	0.089	(L)	5.8e-282	0.1	(L)	0.01587	0.526	(N)
Var	X	[2, 8]	[3, 70]	31.0 ± 38.1	63.3 ± 36.0	60.9 ± 36.4	2.0e-117	2.7e-95	0.276	(M)	2.2e-84	0.289	(M)	0.08552	0.518	(N)
Var	Z	[2, 8]	[3, 70]	1.1 ± 6.0	38.7 ± 36.2	35.8 ± 34.8	1.8e-261	3.1e-236	0.191	(L)	1.4e-220	0.206	(L)	0.05576	0.521	(N)
Var	R_y	[2, 8]	[3, 70]	52.8 ± 33.1	87.0 ± 11.6	87.1 ± 11.7	7.0e-258	4.4e-192	0.173	(L)	9.2e-193	0.172	(L)	0.80537	0.497	(N)
Gs	all	[3, 15]	[5, 17]	55.0 ± 12.8	77.0 ± 9.8	73.0 ± 10.0	1.9e-126	1.2e-103	0.074	(L)	9.2e-83	0.121	(L)	8.4e-09	0.612	(S)
Gs	X	[3, 15]	[5, 17]	30.9 ± 30.3	65.6 ± 27.5	58.3 ± 31.3	2.2e-55	8.6e-51	0.212	(L)	1.2e-32	0.272	(M)	0.00106	0.562	(N)
Gs	Z	[3, 15]	[5, 17]	0.9 ± 5.3	37.3 ± 28.4	27.6 ± 26.4	1.1e-102	5.4e-98	0.136	(L)	5.6e-73	0.205	(L)	4.6e-07	0.594	(S)
Gs	R_y	[3, 15]	[5, 17]	81.1 ± 23.8	94.0 ± 9.2	93.0 ± 10.1	1.4e-12	1.8e-11	0.379	(S)	7.3e-09	0.395	(S)	0.17699	0.524	(N)

5.3 RQ3: Impact of Reduction on QST Effectiveness

Here, we present the results for RQ3a and RQ3b.

5.3.1 RQ3a: Test Effectiveness

In Table 8, we show the summary of the mutation score results, including statistical tests between all the pairs of approaches. The table is divided into five sections for each program category with four rows in each section for the mutant types X, Z and R_y and All, which consists of all three mutant types.

First, we observe that the average mutation score for all program categories and all mutant types is 54.5% for the Default approach, with the reduction approaches Greedy at 74.7% and Random at 74.0%. We see from the statistical tests in the columns **Default-Greedy** and **Default-Random** that there is no difference in killing X mutants overall when applying either reduction approach. However, the mutation score for the Z mutants is close to zero for Default, while 36.0% for Greedy and 33.1% for Random. The R_y mutants are also killed at a higher rate when a reduction is applied, at 90.0% for Greedy and Random, an increase from 67.9% for Default.

By category, we first observe for Grov a statistically significant increase in overall mutation score, but the effect is small, as we can see from the **Default-Greedy** and **Default-Random** columns where the effect size is (S) in favor of the reduction approaches. Interestingly, we observe that Default has a higher mutation score for X mutants at 96.9%, while the reduction approaches obtain scores of 58.7% for Greedy and 62.0% for Random. For R_y we also see an increase when a reduction is applied. Thus, fewer X mutants are killed for Grov when a reduction is applied, but more Z and R_y mutants are killed. For Qwalk and Var, we observe that both achieve a large increase in overall mutation scores for all mutant types. However, the X mutants for Qwalk are killed at a statistically significant higher rate for Default with effect size (s). All other mutants are killed at a higher rate with a large or medium effect size when applying a reduction. Lastly, for the Gs programs we make two distinct observations. First, we see that it obtains a statistically large increase in mutation score when applying either reduction approach, particularly due to the killing of Z mutants. Second, we see from the **Greedy-Random** column, that there are only negligible (N) differences between the Greedy and random reduction approaches, except for the Gs category, which exhibits statistically significant differences overall and for the Z mutants with small effect sizes.

Table 9. Spearman correlation r_s between reduction rate and mutation score. We use N as a shorthand for the correlation magnitude Negligible.

				Correlation r_s		p-value		Correlation Magnitude	
Category	**Mutant Type**	**#Qubits**	**Depth**	**Greedy**	**Random**	**Greedy**	**Random**	**Greedy**	**Random**
All	All	[2, 15]	[3, 684]	0.05689	0.01056	1.9e-04	0.48917	N	N
All	X	[2, 15]	[3, 684]	-0.46222	-0.46551	4.2e-226	1.0e-229	Moderate(-)	Moderate(-)
All	Z	[2, 15]	[3, 684]	0.46749	0.40918	6.3e-232	7.2e-173	Moderate	Moderate
All	R_y	[2, 15]	[3, 684]	0.08801	0.11216	7.7e-09	1.7e-13	N	N
Grov	All	[6, 9]	[30, 684]	-0.00749	-0.03106	0.78099	0.24896	N	N
Grov	X	[6, 9]	[30, 684]	-0.09993	-0.11574	2.0e-04	1.6e-05	N	N
Grov	Z	[6, 9]	[30, 684]	0.12547	0.10869	2.9e-06	5.2e-05	N	N
Grov	R_y	[6, 9]	[30, 684]	-0.03745	-0.03895	0.16435	0.14811	N	N
Qwalk	All	[3, 5]	[14, 154]	-0.05194	-0.02963	0.07576	0.31116	N	N
Qwalk	X	[3, 5]	[14, 154]	-0.63187	-0.61523	2.1e-131	8.8e-123	Moderate(-)	Moderate(-)
Qwalk	Z	[3, 5]	[14, 154]	0.5946	0.58487	8.8e-113	2.6e-108	Moderate	Moderate
Qwalk	R_y	[3, 5]	[14, 154]	-0.00965	-0.01322	0.74169	0.65145	N	N
Var	All	[2, 8]	[3, 70]	-0.18416	-0.19856	1.6e-11	3.3e-13	N	N
Var	X	[2, 8]	[3, 70]	-0.64931	-0.65898	6.9e-159	3.2e-165	Moderate(-)	Moderate(-)
Var	Z	[2, 8]	[3, 70]	0.61412	0.63563	1.1e-137	2.6e-150	Moderate	Moderate
Var	R_y	[2, 8]	[3, 70]	-0.2136	-0.20875	4.4e-15	1.8e-14	Weak(-)	Weak(-)
Gs	All	[3, 15]	[5, 17]	0.30426	0.05494	1.9e-10	0.26125	Weak	N
Gs	X	[3, 15]	[5, 17]	-0.1179	-0.26432	0.01563	3.8e-08	N	Weak(-)
Gs	Z	[3, 15]	[5, 17]	0.17611	-0.03972	2.9e-04	0.41681	N	N
Gs	R_y	[3, 15]	[5, 17]	0.51672	0.44351	4.8e-30	1.1e-21	Moderate	Moderate

5.3.2 RQ3b: Correlation Between Reduction and Effectiveness

Finally, we investigate the correlation between the reduction and approach effectiveness. In Table 9, we show the Spearman correlation between the reduction rate and mutation score by category. For all program categories and mutant types in the first row, we observe a Negligible correlation for the Greedy and Random approaches. For the X mutants, we can see that there is a Moderate negative correlation for both approaches, while the Z mutants have a Moderate positive correlation for both approaches. The X and Z mutant correlation coefficients are similar in absolute values, showing opposite correlations. Finally, the R_y mutants show a Negligible correlation across the program categories. Regarding the individual program categories, we can make two observations. First, for the all categories except Gs, we observe either a Negligible correlation. Thus, Gs is the only program category where we observe a Weak positive correlation for the Greedy approach, while Negligible for the Random. The main contributions to this correlation stem from correlations for the Z and R_y mutants. The second observation is that the X mutants are negatively correlated for all the program categories with significant magnitudes for the correlation coefficients, while the Z mutants have similar magnitude coefficients, but are positively correlated.

RQ3 Summary: The reduction approaches significantly improve mutation scores to 74.7% with the Greedy approach, from 54.5% with the Default approach, primarily driven by an increase in Z and R_y mutant kills. There are no significant differences between the reduction approaches, except in the Gs category, where the Greedy approach shows slightly better performance, mainly due to more effective killing of Z mutants. The overall correlation between reduction effectiveness and mutation score is Negligible for both the approaches.

6 Discussions

Our discussion of the reduction approaches in QST highlights three key themes: (1) **Program Sensitivity**, (2) **Mutant Sensitivity**, and (3) **Greedy vs. Random**. These points guide our discussion on optimizing reduction strategies for diverse QST applications. After these points are addressed, we discuss the practical implications of our findings to the field of QST.

6.1 Program Sensitivity

Although we find that reduction significantly improved both QST efficiency and effectiveness, this is not true for all types of programs.

Higher Depth Programs Benefit More From Reduction? First, we discuss the observation that the programs from Grov experienced a very high runtime improvement from reduction compared to the other programs. Longer-depth circuits take more time to run than short-depth circuits. Thus, by requiring fewer circuit executions due to our approach, we save considerably more *longer* circuit executions, especially for high-depth circuits, such as Grov. This is not always the case, however, as we also find a considerable number of slowdowns in the moderate depth category such as Qwalk, we argue that higher depth alone does not lead to positive benefits in QST. In addition, a high satisfiability of the output criterion is required, such as achieved by the uniform and real amplitudes of the states of Grov and Gs.

Large Program Specifications Benefit More From Reduction: For the programs with large specification sizes (ranks), e.g., Gs, we observe a tendency of both reduction approaches to plateau below a runtime improvement over the Default of 100. This is likely due to the combination of many output states and low depth, which leads to the runtime overhead from the Hadamard gates required for larger reductions competing with the program's depth. To achieve larger reductions, we need more Hadamard gates, which for low depth circuits like Gs programs may cause either a stagnation of runtime improvement or even a slowdown after a certain point. Although our data doesn't provide conclusive answers to either.

Low Satisfiability of Output Criteria. Our approach generally performs poorly for Qwalk programs, aligning with the findings from previous studies indicating that continuous time quantum walks do not typically converge to uniform superpositions [21]. Our results suggest that only programs with specific numbers of walks benefit from reductions, indicating that uniformity in the final state vector is rare. We find some programs in the Var category that obtained no reductions from Algorithm 1. Programs like the quantum conditional phase flip and most integer addition programs failed to achieve any reduction from our approach, suggesting issues with satisfying the output criteria Eq. (6). State vector inspections reveal that this is due to complex amplitudes, meaning they are not eigenstates of mixed Hadamard basis, as our approach requires.

Thus, these are two examples of low satisfiability of the output criteria where reduction is unsuccessful. One where uniformity is not satisfied, and the other where the amplitudes are not real.

6.2 Mutant Sensitivity

We observed that the Default approach hardly detected phase flip faults. By hardly, we mean that this score is minuscule, possibly due to false positives. This is expected as phase flip faults to the final state vector cause flipped signs in the probability amplitudes. Consequently, we cannot discover such Z faults, even if we were to perform an infinite number of measurements in the computational basis.

X Fault vs. Z Fault Detection Tradeoff. While the Z gate acts as a bit flip gate in our reduced basis, allowing phase flip fault discovery, the X gate also modifies its behaviour. We can see from the transformations $X|+\rangle = |+\rangle$ and $X|-\rangle = -|-\rangle$ that X may act as a phase flip gate in the mixed Hadamard basis. Thus, the X and Z gates switch roles, such that in the mixed Hadamard basis, the X gate acts as Z and Z acts as X. This explains why we may observe a reduced detectability of X mutants in our evaluation. The Gs category was the only one that did not experience such an increase, but rather increased detection of X faults. However, we can partially attribute this to the nature of the program category. Due to the large reductions that occur for Gs, there is a smaller number of states in the reduced PS when compared to the other categories' PSs. Thereby, by performing an X gate fault, a basis state not present in the PS is created with a high probability.

6.3 Greedy vs. Random

The Greedy approach can get stuck in local minima, such as $[\mathbb{1}, H, \mathbb{1}]$ demonstrated in our example (see Fig. 3). Considering this, one might expect the Random approach to outperform Greedy when sufficient samples are available to explore the search space of bases. In our experimental setup, we observed that the Greedy approach generally outperformed the Random approach when given the same number of samples, even in systems with fewer qubits.

Curse of Dimensionality The search runtime for Random is comparable to Greedy, both scaling as $O(\#Qubits^2)$, but given the exponential growth of the search space ($2^{\#Qubits}$), Random struggles with the curse of dimensionality [39, 69]. This could limit Random's effectiveness, particularly as the number of qubits increases, as finding a good basis becomes less likely. While increasing the number of random searches might improve the outcomes, the exponentially larger search space makes significant improvements unlikely without incurring substantial runtime overhead.

Some Bases are Only Discovered by Random. The probabilistic nature of the Random approach allows it to occasionally find bases that are not reachable by Greedy, which terminates once no further reductions are identified. This characteristic can be advantageous in scenarios where the search landscape contains multiple viable pathways to a reduction, as shown in **Example A** in the appendix [47]. However, the random approach samples from an exponentially increasing search space of bases. Thus, Random occasionally selects a basis that adds more runtime than it saves due to a high number of Hadamard gates. This is not the case for Greedy, except for some outliers. Greedy, thus, for higher qubit counts of Gs, converges consistently to a basis that results in an improved test runtime over Default while Random is considerably less consistent for high qubit counts.

6.4 Practical Aspects for QST.

Here, we discuss the practical considerations of our approach and results.

Obtaining the Default PS. As our reduction approach assumes that we have access to the state vector of the SUT, serving as a program's "Default PS", we now discuss how we obtain it in practice. We see two approaches to obtain a correct final state vector. The first is through experimental methods such as quantum state tomography [46], which reconstructs the density matrix of a given program. Although full state tomography is resource-intensive and requires exponentially many measurements, more efficient quantum state tomography approaches remain an active research field [7, 11, 16, 44, 65]. Weakly entangled states, such as those we consider, may allow for efficient state tomography using matrix-product state tomography [11, 57]. The second is through mathematical methods of analytical calculation

from probability amplitude expressions. Such as the known transformation formula of quantum Fourier transform or amplitude amplification [46].

Is Greedy Better than Random? While Greedy has an inbuilt stopping criteria for the number of objective function executions, Random requires this as input. The Greedy approach searches more specific paths through the basis search space, while Random may explore more parts of the space. The potential drawbacks of applying the Greedy approach as opposed to Random is: (1) the vulnerability of Greedy to get stuck in local minima, and (2) the inability to find certain bases. We could alleviate (1) through techniques like Basin Hopping [70], incorporating random basis changes to find previously unavailable search paths. While (2), however, could result in the Greedy approach being unable to detect certain faults if the undiscovered basis is required to detect a particular fault. The potential drawbacks of Random as opposed to Greedy are: (1) the curse of dimensionality from exponentially growing search spaces and (2) requiring a specified number of random samples to be input to the algorithm. We can alleviate (2), however, through specifying a formula for the number of searches, such as applying the theoretical runtime formula of the Greedy approach (see appendix [47]).

Thus, while our results show that both Greedy and Random often perform similarly, the Greedy algorithm's tendency to find higher reductions more consistently for higher qubit counts suggests it may be more reliable for practical applications. This is because more qubits, rather than fewer, will be required by future quantum programs to obtain quantum advantage [12, 30].

Reduction-Circuit We see a practical limitation of our approach from how we perform the reduction. While our reduction approach avoids exponential computation of the inverse matrix problem by performing reductions in the reduction circuit, we assume that the circuit can be initialized to the "Default PS". In our experiments, we initialize the state vector directly for experimental simplicity. For a real device, gate instructions create this initialization instead. However, we argue that this threat is mitigated by the same gate instructions that gave us the "Default PS" in the first place. Thus, if we can obtain a "Default PS", we can also initialize it.

7 Related Work

This section discusses related work.

Abstraction of Quantum Circuits To harness the benefits of quantum algorithms, efficient methods for designing and automating software are crucial in quantum software engineering. The work of Wille et al. [74] tackles this by abstracting quantum concepts, such as circuits and state vectors, using decision diagrams to simplify quantum design automation. Similarly, our reduced program specification advances the abstraction of quantum program specifications in QST by introducing basis dependence. Although our approach is complemented by the high level abstractions from Wille et al. [74], our experimental evaluation demonstrates how these abstract concepts can practically improve QST.

Projective Measurements In the work of Li et al. [37], projective measurements are applied by runtime assertions to obtain more efficient and effective testing. They provide an implementation strategy for their assertions along with a case study of realistic quantum algorithms such as Shor's algorithm and HHL. While our projective measurements consider mixed Hadamard bases, similar to their superposition assertions, they also include projective measurements for validating maximally entangled states, such as with the Bell basis. In their approach, knowledge about the quantum algorithms is utilized to obtain their initial projections. Similarly, we do the same to obtain our "Default PS", which we in turn utilize through our approach to obtain our projective measurements. Thus, we view our approach as complimentary

on the theoretical side, offering a similar approach, but we include program specifications as a high level software testing component to guide QST on the implementation side. In addition, our experimental evaluation provides empirical evidence of the success of projective measurements in general for more efficient and effective QST.

Other QST Methodologies In quantum symbolic execution, Nan et al. [45] exploit quantum superposition to provide multiple inputs to a quantum program test SUT, effectively reducing the quantum resource requirements. Although their approach does not detect end-circuit phase flip faults, as X basis measurements are required, they obtain a large input coverage. We only use a single input for each of our study subjects, focusing on a larger coverage on the output side.

Another approach by Paltenghi and Pradel [49] based on metamorphic testing utilizes metamorphic relations by comparing the statistical distributions between a source and a transformed follow-up quantum program. Some benefits of this approach is that it allows for a more high level handling of programming bugs on the language specific statement level, in addition to requiring less knowledge about the quantum algorithm making it easily applicable. However, in order to detect circuit faults efficiently and effectively, especially phase flip faults, a metamorphic relationship with quantum-specific algorithmic knowledge could be required. By employing projective measurements such as in our approach or in Li et al. [37], we may alleviate both efficiency of sampling and detection of phase flip faults.

8 Conclusions and Future Directions

In the growing field of quantum computing, efficient and effective quantum software testing is essential. To this aim, we proposed an approach to reduce quantum program specifications to perform projective measurements in mixed Hadamard bases. We empirically evaluated our approach and found that reduction is highly efficient and effective, with the Greedy approach outperforming the Random baseline in terms of average reduction rate and runtime. Specifically, reductions were most effective in the Grover Search (Grov) and Graph States (Gs) categories, demonstrating high and consistent reduction rates, whereas Quantum Walk (Qwalk) and Various (Var) categories showed significantly poorer performance.

Regarding the impact on quantum software testing (QST) efficiency, our reduction approach enhanced testing efficiency compared to the Default approach where only computational basis measurements are performed, with the Greedy method slightly more favorable in Gs programs for larger specifications. The greatest improvements were noted in Grov and Gs categories, highlighting the influence of program characteristics on reduction success. However, the correlation between the degree of reduction and actual runtime improvements was Moderate, indicating that reductions do not uniformly predict efficiency improvements.

Regarding the effectiveness of QST, applying reduced specifications significantly improved mutation scores, particularly through enhanced detection of phase flip faults. Although the performance between reduction methods was generally comparable, the Greedy approach showed a slight advantage in the Gs category. The overall correlation between reduction and mutation score effectiveness was Negligible overall, driven by program characteristics and the inverse relationship of detecting X and Z faults. These results underline the potential and limitations of reduction approaches in enhancing the efficiency and effectiveness of QST, providing valuable insights into the dependency of performance gains on specific program characteristics and reduction strategies.

Going forward, future research should explore two promising areas. The first involves developing methods to obtain program specifications empirically or mathematically to streamline the reduction process. The second area focuses on

generalizing the reduction algorithm using advanced search techniques like genetic algorithms, further extending the types of projective measurements our approach can support.

Acknowledgments

This work is supported by the Qu-Test project (299827) funded by the Research Council of Norway (RCN) and has benefited from the Experimental Infrastructure for Exploration of Exascale Computing (eX^3), which is supported by the RCN project 270053. S. Ali also acknowledges the support from Simula's internal strategic project on quantum software engineering and the *Quantum Hub initiative* (OsloMet).

References

[1] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, et al. 2019. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2562111

[2] Shaukat Ali and Tao Yue. 2023. Quantum Software Testing: A Brief Introduction. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. 332–333. https://doi.org/10.1109/ICSE-Companion58688.2023.00093

[3] Simon Anders and Hans J. Briegel. 2006. Fast simulation of stabilizer circuits using a graph-state representation. *Phys. Rev. A* 73 (Feb 2006), 022334. Issue 2. https://doi.org/10.1103/PhysRevA.73.022334

[4] Mohamed Raed El aoun, Heng Li, Foutse Khomh, and Lionel Tidjon. 2022. Bug Characteristics in Quantum Software Ecosystem. (2022). arXiv:2204.11965 [cs.SE]

[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* (Waikiki, Honolulu, HI, USA) *(ICSE '11)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/1985793.1985795

[6] Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: a critical review and guidelines. *Empirical Softw. Engg.* 27, 4 (jul 2022), 31 pages. https://doi.org/10.1007/s10664-021-10072-8

[7] T Baumgratz, A Nüßeler, M Cramer, and M B Plenio. 2013. A scalable maximum likelihood method for quantum state tomography. *New Journal of Physics* 15, 12 (Dec. 2013), 125004. https://doi.org/10.1088/1367-2630/15/12/125004

[8] Zhenyu Cai, Ryan Babbush, Simon C. Benjamin, Suguru Endo, William J. Huggins, Ying Li, et al. 2023. Quantum error mitigation. *Reviews of Modern Physics* 95, 4 (Dec. 2023). https://doi.org/10.1103/revmodphys.95.045005

[9] J. Campos and A. Souto. 2021. QBugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*. IEEE Computer Society, Los Alamitos, CA, USA, 28–32. https://doi.org/10.1109/Q-SE52541.2021.00013

[10] Joshua Combes, Kyle V. Gulshen, and Matthew P. Harrigan et al. 2019. *Forest Benchmarking: QCVV using PyQuil*. https://doi.org/10.5281/zenodo.3455848

[11] Marcus Cramer, Martin B. Plenio, Steven T. Flammia, Rolando Somma, David Gross, Stephen D. Bartlett, et al. 2010. Efficient quantum state tomography. *Nature Communications* 1, 1 (dec 2010). https://doi.org/10.1038/ncomms1147

[12] Alexander M. Dalzell, Aram W. Harrow, Dax Enshan Koh, and Rolando L. La Placa. 2020. How many qubits are needed for quantum computational supremacy? *Quantum* 4 (May 2020), 264. https://doi.org/10.22331/q-2020-05-11-264

[13] R.A. DeMillo, R.J. Lipton, and F.G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer* 11, 4 (1978), 34–41. https://doi.org/10.1109/C-M.1978.218136

[14] Cirq Developers. 2023. *Cirq*. https://doi.org/10.5281/zenodo.10247207

[15] Daniel J. Egger, Claudio Gambella, Jakub Marecek, Scott McFaddin, Martin Mevissen, Rudy Raymond, et al. 2020. Quantum Computing for Finance: State-of-the-Art and Future Prospects. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–24. https://doi.org/10.1109/tqe.2020.3030314

[16] Jens Eisert, Dominik Hangleiter, Nathan Walk, Ingo Roth, Damian Markham, Rhea Parekh, et al. 2020. Quantum certification and benchmarking. *Nature Reviews Physics* 2, 7 (June 2020), 382–390. https://doi.org/10.1038/s42254-020-0186-4

[17] Pablo Fernández and Miguel A. Martin-Delgado. 2024. Implementing the Grover Algorithm in Homomorphic Encryption Schemes. arXiv:2403.04922 [quant-ph]

[18] Richard P. Feynman. 1982. Simulating physics with computers. *International Journal of Theoretical Physics* 21, 6 (June 1982), 467–488. https://doi.org/10.1007/BF02650179

[19] 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* (Virtual, South Korea,) *(ISSTA 2022)*. Association for Computing Machinery, New York, NY, USA, 797–800. https://doi.org/10.1145/3533767.3543296

[20] Guillermo García-Pérez, Matteo A.C. Rossi, Boris Sokolov, Francesco Tacchino, Panagiotis Kl. Barkoutsos, Guglielmo Mazzola, et al. 2021. Learning to Measure: Adaptive Informationally Complete Generalized Measurements for Quantum Algorithms. *PRX Quantum* 2, 4 (Nov. 2021). https:

//doi.org/10.1103/prxquantum.2.040342

[21] Heath Gerhardt and John Watrous. 2003. Continuous-Time Quantum Walks on the Symmetric Group. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 290–301. https://doi.org/10.1007/978-3-540-45198-3_25

[22] Dale Goodhue, William Lewis, and Ron Thompson. 2012. Research note: Does PLS have advantages for small sample size or non-normal data? *MIS Quarterly* 36 (09 2012), 981–1001. https://doi.org/10.2307/41703490

[23] Matteo Gori, Matthieu Sarkis, and Alexandre Tkatchenko. 2024. Gaussian Entanglement Measure: Applications to Multipartite Entanglement of Graph States and Bosonic Field Theory. arXiv:2401.17938 [quant-ph]

[24] Lov K. Grover. 1996. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Philadelphia, Pennsylvania, USA) *(STOC '96)*. Association for Computing Machinery, New York, NY, USA, 212–219. https://doi.org/10.1145/237814.237866

[25] Leonid Gurvits. 2003. Classical deterministic complexity of Edmonds' Problem and quantum entanglement. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (San Diego, CA, USA) *(STOC '03)*. Association for Computing Machinery, New York, NY, USA, 10–19. https://doi.org/10.1145/780542.780545

[26] M. Hein, J. Eisert, and H. J. Briegel. 2004. Multiparty entanglement in graph states. *Physical Review A* 69, 6 (jun 2004). https://doi.org/10.1103/physreva.69.062311

[27] Melinda Hess and Jeffrey Kromrey. 2004. Robust Confidence Intervals for Effect Sizes: A Comparative Study of Cohen's d and Cliff's Delta Under Non-normality and Heterogeneous Variances. *Paper Presented at the Annual Meeting of the American Educational Research Association* (01 2004).

[28] 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* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 430–435. https://doi.org/10.1145/3387940.3391459

[29] Hsin-Yuan Huang, Michael Broughton, Jordan Cotler, Sitan Chen, Jerry Li, Masoud Mohseni, et al. 2022. Quantum advantage in learning from experiments. *Science* 376, 6598 (2022), 1182–1186. https://doi.org/10.1126/science.abn7293

[30] Tsubasa Ichikawa, Hideaki Hakoshima, Koji Inui, Kosuke Ito, Ryo Matsuda, Kosuke Mitarai, et al. 2023. A comprehensive survey on quantum computer usage: How many qubits are employed for what purposes? arXiv:2307.16130 [quant-ph]

[31] Christian M. Ringle Joe F. Hair and Marko Sarstedt. 2011. PLS-SEM: Indeed a Silver Bullet. *Journal of Marketing Theory and Practice* 19, 2 (2011), 139–152. https://doi.org/10.2753/MTP1069-6679190202 arXiv:https://doi.org/10.2753/MTP1069-6679190202

[32] E.R. Johnston, N. Harrigan, and M. Gimeno-Segovia. 2019. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly Media, Incorporated.

[33] Thierry Nicolas Kaldenbach and Matthias Heller. 2023. Mapping quantum circuits to shallow-depth measurement patterns based on graph states. arXiv:2311.16223 [quant-ph]

[34] K. Khadiev and E. Krendeleva. 2023. Quantum Algorithm for Searching of Two Sets Intersection. *Russian Microelectronics* 52, S1 (Dec. 2023), S379–S383. https://doi.org/10.1134/s106373972360084x

[35] William H. Kruskal and W. Allen Wallis. 1952. Use of Ranks in One-Criterion Variance Analysis. *J. Amer. Statist. Assoc.* 47, 260 (1952), 583–621. https://doi.org/10.1080/01621459.1952.10483441

[36] Christoph Laaber, Tao Yue, and Shaukat Ali. 2024. Evaluating Search-Based Software Microbenchmark Prioritization. *IEEE Transactions on Software Engineering* (March 2024), 1–16. https://doi.org/10.1109/TSE.2024.3380836

[37] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs. *Proc. ACM Program. Lang.* 4, OOPSLA, Article 150 (Nov. 2020), 29 pages. https://doi.org/10.1145/3428218

[38] X. X. Li, D. X. Li, and X. Q. Shao. 2024. Generation of complete graph states in a spin-1/2 Heisenberg chain with a globally optimized magnetic field. *Physical Review A* 109, 4 (April 2024). https://doi.org/10.1103/physreva.109.042604

[39] R. B. MARIMONT and M. B. SHAPIRO. 1979. Nearest Neighbour Searches and the Curse of Dimensionality. *IMA Journal of Applied Mathematics* 24, 1 (08 1979), 59–70. https://doi.org/10.1093/imamat/24.1.59

[40] K. McInroy, N. Pearson, and J. D. Pritchard. 2024. Benchmarking the algorithmic performance of near-term neutral atom processors. arXiv:2402.02127 [quant-ph]

[41] Eñaut Mendiluze, Shaukat Ali, Paolo Arcaini, and Tao Yue. 2021. Muskit: A Mutation Analysis Tool for Quantum Software Testing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1266–1270. https://doi.org/10.1109/ASE51524.2021.9678563

[42] 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* (Seoul, South Korea) *(ICSE-NIER '20)*. Association for Computing Machinery, New York, NY, USA, 29–32. https://doi.org/10.1145/3377816.3381731

[43] Nikolaj Moll, Panagiotis Barkoutsos, Lev S Bishop, Jerry M. Chow, Andrew Cross, Daniel J. Egger, et al. 2018. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology* 3, 3 (2018), 030503. https://doi.org/10.1088/2058-9565/aab822

[44] L. Motka, B. Stoklasa, J. Rehacek, Z. Hradil, V. Karasek, D. Mogilevtsev, et al. 2014. Efficient algorithm for optimizing data-pattern tomography. *Physical Review A* 89, 5 (May 2014). https://doi.org/10.1103/physreva.89.054102

[45] Jiang Nan, Wang Zichen, and Wang Jian. 2023. Quantum symbolic execution. *Quantum Information Processing* 22, 10 (20 Oct. 2023), 389. https://doi.org/10.1007/s11128-023-04144-5

[46] Michael A. Nielsen and Isaac L. Chuang. 2011. *Quantum Computation and Quantum Information: 10th Anniversary Edition* (10th ed.). Cambridge University Press, USA.

[47] Noah D. Oldfield, Christoph Laaber, Tao Yue, and Shaukat Ali. 2024. *Replication package for our paper entitled "Faster and Better Quantum Software Testing through Specification Reduction and Projective Measurements".* https://doi.org/10.5281/zenodo.11191215

[48] Matteo Paltenghi and Michael Pradel. 2022. Bugs in Quantum computing platforms: an empirical study. *Proc. ACM Program. Lang.* 6, OOPSLA1, Article 86 (apr 2022), 27 pages. https://doi.org/10.1145/3527330

[49] Matteo Paltenghi and Michael Pradel. 2023. MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 2413–2424. https://doi.org/10.1109/ICSE48619.2023.00202

[50] Karl Pearson. 1900. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50, 302 (July 1900), 157–175. https://doi.org/10.1080/14786440009463897

[51] Hugo Pillin, Gilles Burel, Paul Baird, El-Houssain Baghious, and Roland Gautier. 2023. Hypercube quantum search: exact computation of the probability of success in polynomial time. *Quantum Information Processing* 22, 3 (March 2023). https://doi.org/10.1007/s11128-023-03883-9

[52] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79

[53] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. 2023. MQT Bench: Benchmarking Software and Design Automation Tools for Quantum Computing. *Quantum* (2023). MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/.

[54] Eleanor Rieffel and Wolfgang Polak. 2011. *Quantum Computing: A Gentle Introduction* (1st ed.). The MIT Press.

[55] Jun John Sakurai. 1994. *Modern quantum mechanics; rev. ed.* Addison-Wesley, Reading, MA. https://cds.cern.ch/record/1167961

[56] Raqueline A. M. Santos. 2016. Szegedy's quantum walk with queries. *Quantum Information Processing* 15, 11 (1 Nov. 2016), 4461–4475. https://doi.org/10.1007/s11128-016-1427-4

[57] Tobias Schmale, Moritz Reh, and Martin Gärttner. 2022. Efficient quantum state tomography with convolutional neural networks. *npj Quantum Information* 8, 1 (Sept. 2022). https://doi.org/10.1038/s41534-022-00621-4

[58] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients: appropriate use and interpretation. *Anesthesia & analgesia* 126, 5 (2018), 1763–1768. https://doi.org/10.1213/ANE.0000000000002864

[59] Raphael Seidel, Colin Kai-Uwe Becker, Sebastian Bock, Nikolay Tcholtchev, Ilie-Daniel Gheorghe-Pop, and Manfred Hauswirth. 2023. Automatic generation of Grover quantum oracles for arbitrary data structures. *Quantum Science and Technology* 8, 2 (2023), 025003. https://doi.org/10.1088/2058-9565/acaf9d

[60] Peter W. Shor. 1995. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A* 52 (Oct 1995), R2493–R2496. Issue 4. https://doi.org/10.1103/PhysRevA.52.R2493

[61] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (oct 1997), 1484–1509. https://doi.org/10.1137/s0097539795293172

[62] Nikolai A. Sinitsyn and Bin Yan. 2023. Topologically protected Grover's oracle for the partition problem. *Physical Review A* 108, 2 (Aug. 2023). https://doi.org/10.1103/physreva.108.022412

[63] Dag I. K. Sjøberg and Gunnar Rye Bergersen. 2023. Construct Validity in Software Engineering. *IEEE Transactions on Software Engineering* 49, 3 (2023), 1374–1396. https://doi.org/10.1109/TSE.2022.3176725

[64] Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of Software Engineering Research. *ACM Trans. Softw. Eng. Methodol.* 27, 3, Article 11 (sep 2018), 51 pages. https://doi.org/10.1145/3241743

[65] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. 2018. Neural-network quantum state tomography. *Nature Physics* 14, 5 (Feb. 2018), 447–450. https://doi.org/10.1038/s41567-018-0048-5

[66] Tim van Leent, Matthias Bock, Florian Fertig, Robert Garthoff, Sebastian Eppelt, Yiru Zhou, et al. 2022. Entangling single atoms over 33 km telecom fibre. *Nature* 607, 7917 (jul 2022), 69–73. https://doi.org/10.1038/s41586-022-04764-4

[67] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132. https://doi.org/10.3102/10769986025002101

[68] Salvador Elías Venegas-Andraca. 2012. Quantum walks: a comprehensive review. *Quantum Information Processing* 11, 5 (July 2012), 1015–1106. https://doi.org/10.1007/s11128-012-0432-5

[69] Nico Vervliet, Otto Debals, Laurent Sorber, and Lieven De Lathauwer. 2014. Breaking the Curse of Dimensionality Using Decompositions of Incomplete Tensors: Tensor-based scientific computing in big data analysis. *IEEE Signal Processing Magazine* 31, 5 (2014), 71–79. https://doi.org/10.1109/MSP.2014.2329429

[70] David J. Wales and Jonathan P. K. Doye. 1997. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *The Journal of Physical Chemistry A* 101, 28 (1997), 5111–5116. https://doi.org/10.1021/jp970984n

[71] Jiyuan Wang, Qian Zhang, Guoqing Harry Xu, and Miryung Kim. 2021. QDiff: Differential Testing of Quantum Software Stacks. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, Australia, 692–704. https://doi.org/10.1109/ASE51524.2021.9678792

[72] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2021. Quito: a Coverage-Guided Test Generator for Quantum Programs. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1237–1241. https://doi.org/10.1109/ASE51524.2021.9678798

[73] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. QuSBT: Search-Based Testing of Quantum Programs. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 173–177. https://doi.org/10.1145/3510454.3516839

[74] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. 2022. Decision diagrams for quantum computing. In *Design Automation of Quantum Computers*. Springer, 1–23.

[75] Mingyou Wu. 2024. Efficiency of k-Local Quantum Search and its Adiabatic Variant on Random k-SAT. arXiv:2403.03237 [quant-ph]

[76] Jianjun Zhao. 2021. Quantum Software Engineering: Landscapes and Horizons. arXiv:2007.07047 [cs.SE]

[77] P. Zhao, J. Zhao, Z. Miao, and S. Lan. 2021. Bugs4Q: A Benchmark of Real Bugs for Quantum Programs. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE Computer Society, Los Alamitos, CA, USA, 1373–1376. https://doi.org/10.1109/ASE51524.2021.9678908

[78] Li Zhou, Nengkun Yu, and Mingsheng Ying. 2019. An applied quantum Hoare logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Phoenix, AZ, USA) *(PLDI 2019)*. Association for Computing Machinery, New York, NY, USA, 1149–1162. https://doi.org/10.1145/3314221.3314584