Dynamic Transitive Closure-Based Static Analysis through the Lens of Quantum Search

JIAWEI REN, University of New South Wales, Australia
YULEI SUI, University of New South Wales, Australia
XIAO CHENG, University of New South Wales, Australia
YUAN FENG, University of Technology Sydney, Australia
JIANJUN ZHAO, Kyushu University, Japan

Many existing static analysis algorithms suffer from cubic bottlenecks because of the need to compute a dynamic transitive closure (DTC). For the first time, this paper studies the quantum speedups on searching subtasks in DTC-based static analysis algorithms using quantum search (e.g., Grover's algorithm). We first introduce our oracle implementation in Grover's algorithm for DTC-based static analysis and illustrate our quantum search subroutine. Then, we take two typical DTC-based analysis algorithms: context-free-language reachability and set constraint-based analysis, and show that our quantum approach can reduce the time complexity of these two algorithms to truly subcubic ($O(N^2\sqrt{N}polylog(N))$), yielding better results than the upper bound ($O(N^3/\log N)$) of existing classical algorithms. Finally, we conducted a classical simulation of Grover's search to validate our theoretical approach, due to the current quantum hardware limitation of lacking a practical, large-scale, noise-free quantum machine. We evaluated the correctness and efficiency of our approach using IBM Qiskit on nine open-source projects and randomly generated edge-labeled graphs/constraints. The results demonstrate the effectiveness of our approach and shed light on the promising direction of applying quantum algorithms to address the general challenges in static analysis.

CCS Concepts: • **Computer systems organization → Quantum computing**; • **Software and its engineering → Automated static analysis**.

Additional Key Words and Phrases: CFL-reachability, Set constraint-based analysis, Grover's search

1 INTRODUCTION

Static analysis plays a crucial role in approximating the runtime behavior of programs without the need for actual execution, enabling a wide range of applications such as program optimization [10, 26], software vulnerability detection [24, 49, 55, 56, 60] and code embedding [16–18, 53]. However, a cubic bottleneck [34] has appeared as a common issue in various static analysis algorithms, such as context-free-language (CFL) reachability [37, 38, 42], recursive state machine (RSM) reachability [15], and set constraint-based analysis [3, 33]. Overcoming this cubic time complexity is a significant but highly challenging task.

Existing Efforts and Challenges. To better understand the cubic bottleneck, Melski and Reps [42] relate the data-flow reachability to the problem of CFL-reachability. Later, Heintze

Authors' addresses: Jiawei Ren, University of New South Wales, Sydney, Australia, jiawei.ren@student.unsw.edu.au; Yulei Sui, University of New South Wales, Sydney, Australia, y.sui@unsw.edu.au; Xiao Cheng, University of New South Wales, Sydney, Australia, xiao.cheng@unsw.edu.au; Yuan Feng, University of Technology Sydney, Sydney, Australia, yuan.feng@uts.edu.au; Jianjun Zhao, Kyushu University, Fukuoka, Japan, zhao@ait.kyushu-u.ac.jp.

ACM 1049-331X/2024/1-ART1
https://doi.org/10.1145/3644389

and McAllestert [34] formulate the subtyping and data-flow reachability problems as 2-way nondeterministic pushdown automata (2NPDA [2]) and prove that they are 2NPDA-complete such that it is difficult to find a subcubic solution. Previous studies [15, 42] have shown that many of these static analysis approaches are inter-convertible (e.g., CFL-reachability and set constraint-based analysis) and even equivalent (e.g., CFL-reachability and RSM-reachability). This observation highlights the shared intrinsic structures that contribute to the cubic bottleneck.

The primary reason for the $O(N^3)$ bottleneck is that these algorithms are based on *dynamic transitive closure* (DTC) [34, 36, 48] — maintaining the reachability information in a directed graph between arbitrary pairs of vertices while the graph is dynamically changing [31]. DTC-based static analysis algorithms are partially dynamic and incremental, i.e., only vertex/edge insertions occur without deletions. The *searching subtask* in a DTC-based analysis is a repeated procedure aiming to find new reachable information for each vertex/edge. It occupies a significant portion of the total analysis time to ensure the convergence and soundness of the underlying algorithm. Taking the CFL-reachability algorithm as an example, given N vertices in an input graph, the algorithm stores the initial edges in a worklist and performs the $O(N)$-time searching subtask for each edge in the worklist to find new reachable vertices. As the worklist is dynamically updated until convergence, the repetition of searching subtasks is bounded by the total number of edges which is $O(N^2)$, resulting in $O(N^3)$ as the total complexity.

It is challenging to break through this cubic bottleneck for general DTC-based analysis. The existing efforts to improve the efficiency of DTC-based static analysis fall into two categories. One is to reduce the complexity without sacrificing precision. Chaudhuri [15] improves the searching subtasks of CFL-reachability by adapting fast set operations, reducing the total complexity to $O(N^3/\log N)$ as the best classical upper bound, which achieves a logarithmic factor speedup over cubic. Boolean Matrix Multiplication (BMM) implies that the transitive closure problem can be solved in truly subcubic time [61], but it works for a restricted version of DTC-based static analysis. An algorithm is truly subcubic if it runs in $O(N^{3-\epsilon})$ time for some $\epsilon > 0$. For instance, Chatterjee et al. show that solving Dyck-CFL-reachability, a restricted version of CFL-reachability working on the Dyck language in general graphs, is BMM hard [14]. However, the truly subcubic $O(N^{3-\omega})$ bound of CFL-reachability cannot be obtained without obtaining $O(N^{3-\omega})$-time combinatorial algorithms for BMM [20, 59].

Another way to speed up DTC-based analysis is to trade precision for efficiency or solve a subset of DTC-based problems with reduced complexity. Taking points-to analysis as an example, Steensgaard's conditional unification approach [51] achieves an almost linear complexity but yields imprecise results compared to inclusion-based Anderson's analysis [8]. As another example, Dyck-CFL-reachability [64] can be solved using bidirected trees and graphs in linear time. However, the approach only works on a special case (i.e., the Dyck language) of CFL-reachability analysis.

Insights and Objectives. Pushing the limit of classical algorithms to reduce the complexity of general DTC-based static analysis is extremely hard, with no truly subcubic solution (the best upper bound is $O(N^3/\log N)$) for decades. This work seeks inspiration from an emerging field: quantum computing, which has been well-progressed during the past few years and has shown promising results for many applications. However, applying quantum techniques to address classical problems in the area of programming languages (e.g., cubic bottleneck in static analysis) is still an open question. Our study found that quantum search with a well-designed oracle can solve classical program analysis problems more efficiently. This paper makes the first step by proposing a new quantum search oracle for fundamental DTC-based static analyses and proving that breaking through the cubic bottleneck to truly subcubic is feasible.

Grover's algorithm or Grover's search [30] offers a quadratic speedup on unsorted search problems, whereas the classical approach needs a linear time. A few applications have applied

Grover's search to classical searching problems. In particular, traditional graph searching can be improved using Grover's algorithm. For example, Dürr et al. [23] present the quantum speedups for minimum spanning tree, connectivity, strong connectivity, and single-source shortest path problems. Ambainis and Špalek [7] propose a quantum breadth-first search algorithm. The key to the effectiveness (in terms of efficiency and consistency with classical results) of Grover's search lies in its oracle design and implementation. In this work, we aim to explore a highly efficient oracle that works with probability amplification as an alternative subroutine for searching subtasks in DTC-based static analysis to reduce their total complexity.

Our Solution. For the first time, we present the quantum speedups on DTC-based static analysis algorithms in this paper. Our approach involves utilizing Grover's algorithm to handle the core searching subtasks within dynamic transitive closures. However, applying Grover's algorithm as a subroutine to solve classical problems is not straightforward. Because quantum algorithms operate on quantum inputs, we must first encode classical information into quantum states (or encode classical functions into quantum oracles, as required by Grover's search) to solve static analysis problems, which is a non-trivial task; well-thought-out designs must be invented to avoid the exponential slowdown in the encoding process. To address this, we demonstrate the use of QRAM [29] to load classical data into a quantum superposition, enabling consistent results with classical algorithms while dealing with the I/O problem in DTC-based static analysis. In addition to input encoding and oracle design, solving (deterministic) static analysis problems using quantum subroutines also requires taming the probabilistic nature of quantum computing. Quantum algorithms are inherently probabilistic [11], and with some small probability, Grover's search may return inconsistent results compared to classical methods. To reconcile the deterministic nature of static analysis with the probabilistic nature of quantum algorithms, we apply a probability amplification technique [12] (because each output of Grover's search can be verified in constant time) to reduce the error probability to a desirable level to produce consistent outputs.

We take two typical DTC-based algorithms, CFL-reachability [48] and set constraint-based analysis [33], as our two client applications. As acknowledged by previous efforts, it is hard to find a subcubic procedure for DTC-based static analysis [34]. We address this challenge by presenting a general approach to reducing the overhead of key searching subtasks, which contributes most to the cubic bottleneck and can be replaced by our quantum search subroutine rather than changing the entire structure of the algorithms. For instance, when searching for new reachable vertices of a given edge in an edge-labeled graph during CFL-reachability analysis, the time complexity is reduced from linear to square root using our search subroutine. The total cost is thus reduced, contributed by the cost reduction in each searching subtask. We prove that the (worst-case expected) time complexity of our approach yields a truly subcubic result $O(N^{2.5}polylog(N))$ (the exponent of $polylog(N)$ is two) rather than the best-known upper bound by the classical solution. It's important to note that current limitations in quantum hardware present obstacles to the practical use of Grover's search today. Therefore, we need to clarify two general assumptions prevalent in the quantum community: the availability of large-scale, noise-free quantum machines and the QRAM model, both of which are discussed in detail in Section 7. It is essential to understand that these assumptions confine the primary contribution of this paper to a theoretical improvement in the time complexity of the cubic bottleneck in DTC-based static analysis, given the current state of technology.

In summary, we make the following contributions:

- We propose the first truly subcubic complexity solution for DTC-based static analysis algorithms equipped with quantum subroutines. The time complexity of two applications: CFL-reachability (Theorem 4.2) and set constraint-based analysis (Theorem 5.2), is improved

to $O(N^{2.5}polylog(N))$ by replacing classical exhaustive search with the quantum search subroutine to reduce the overall overhead.
- To apply the quantum search subroutine, we present an efficient oracle implementation of Grover's search based on the QRAM model [29] (Section 3.2), which supports loading classical data into quantum superposition in $O(polylog(N))$ time, hence making the algorithm practical without affecting the actual speedup. To ensure consistent results with those by the classical approaches, we employ a probability amplification technique to tame further the probabilistic nature of the quantum search (Section 3.3). We formulate the quantum search subroutine for DTC-based static analysis in Algorithm 3 and show that our quantum subroutine produces consistent outputs with classical counterparts.
- We evaluate both the efficiency and correctness of our approach compared to classical methods. We use nine real-world programs and randomly generated edge-labeled graphs/constraints for evaluation using estimations and simulations based on IBM Qiskit [4] (Section 6).

2 BACKGROUND ON QUANTUM COMPUTING

This section provides a brief background of quantum computing and Grover's search, which serve as the preliminaries of our approach.

2.1 Quantum Bit and Quantum Measurement

In classical computation and information, the bit is the fundamental concept, which can be either 0 or 1 at a time. The *quantum bit (qubit)* is an analogous concept for quantum computation and information. Quantum mechanics use Dirac notation, and a state x in classical computing is denoted as $|x\rangle$ in quantum computing, pronounced "ket x." A qubit state can be $|0\rangle$ or $|1\rangle$, like classical computing. It can also be a *superposition* $|\phi\rangle$, the linear combination of states $|0\rangle$ and $|1\rangle$, written as: $|\phi\rangle = a_0\,|0\rangle + a_1\,|1\rangle$, where a_0 and a_1 are complex numbers and $|a_0|^2 + |a_1|^2 = 1$. The superposition of an n-qubit system is the linear combination of *all possible states of n classical bits* (which we call computational basis states in this paper), written as: $|\phi\rangle = \sum_{i=0}^{2^n-1} a_i\,|i\rangle$, where a_i are complex numbers and $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$.

The advantage of a quantum system is its ability to be in multiple computational basis states at the same time. Specifically, a classical bit can be in either 0 or 1 at a time, but a qubit can be in a superposition, representing $|0\rangle$ and $|1\rangle$ at the same time. For an n-qubit system, a quantum system can represent 2^n computational basis states at the same time, which offers more power on some problems than classical systems.

Quantum measurement is the way to observe the state of qubits. Suppose we have an arbitrary superposition $|\phi\rangle = \sum_{i=0}^{2^n-1} a_i\,|i\rangle$; we can measure $|\phi\rangle$ to get its state. However, we cannot see what exact superposition it is. We can only get a computational basis state i after measurement. The measurement operation on qubits makes the superposition state collapse to one of the computational basis states $|i\rangle$ with probability $|a_i|^2$.

2.2 Quantum Gate

The quantum circuit model [19] is developed to enable quantum parallelism on qubits. The basic operations of the circuit model are called quantum gates. There are two categories: single-qubit gates and multi-qubit gates.

Single-qubit gates are operations on a single qubit. They turn a qubit state into another state. Here, we discuss *X gate* and *Hadamard gate*.

X gate, also called a quantum NOT gate, performs logical NOT in quantum computing. When applying the X gate on the two computational basis states, the state turns from $|0\rangle$ and $|1\rangle$ to $|1\rangle$

and $|0\rangle$, respectively. For an arbitrary superposition, it swaps the coefficient of computational basis states:

$$X(a_0 |0\rangle + a_1 |1\rangle) = a_0 |1\rangle + a_1 |0\rangle$$

Hadamard gate is another typical gate. In general, the effect of the Hadamard gate is:

$$H(a_0 |0\rangle + a_1 |1\rangle) = \frac{a_0 + a_1}{\sqrt{2}} |0\rangle + \frac{a_0 - a_1}{\sqrt{2}} |1\rangle$$

Hadamard gate has a specific usage when applied on $|0\rangle$ and $|1\rangle$ to generate a superposition with an equal probability of observing 0 or 1, which is widely used in algorithms such as Grover's search:

$$H(|0\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$$

$$H(|1\rangle) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle$$

Multi-qubit gates work on two or more qubits. We show the *CNOT gate* and *Toffoli gate*.

The CNOT gate, also called the CX gate, operates on two qubits. The effect is to flip the state of the target qubit if and only if the control qubit is $|1\rangle$. Suppose the first qubit is the control qubit and the second qubit is the target qubit, then:

$$\begin{aligned} &CNOT(a_0 |00\rangle + a_1 |01\rangle + a_2 |10\rangle + a_3 |11\rangle) = \\ &a_0 |00\rangle + a_1 |01\rangle + a_2 |11\rangle + a_3 |10\rangle \end{aligned}$$

The Toffoli gate is a gate that works on three qubits: two control qubits and one target qubit. The effect is to apply X gate on the target qubit if and only if both control qubits are $|1\rangle$. Suppose the first two qubits are control qubits and the third qubit is the target qubit, then:

$$\begin{aligned} &Toffoli(a_0 |000\rangle + a_1 |001\rangle + a_2 |010\rangle + a_3 |011\rangle + \\ &a_4 |100\rangle + a_5 |101\rangle + a_6 |110\rangle + a_7 |111\rangle) = \\ &a_0 |000\rangle + a_1 |001\rangle + a_2 |010\rangle + a_3 |011\rangle + \\ &a_4 |100\rangle + a_5 |101\rangle + a_6 |111\rangle + a_7 |110\rangle \end{aligned}$$

Especially when setting the target qubit to $|0\rangle$, the effect is the same as computing logical AND (used in the implementation section) of two control qubits and storing the result in the target qubit:

$$\begin{aligned} &Toffoli(a_0 |000\rangle + a_1 |010\rangle + a_2 |100\rangle + a_3 |110\rangle) = \\ &a_0 |000\rangle + a_1 |010\rangle + a_2 |100\rangle + a_3 |111\rangle \end{aligned}$$

2.3 Quantum Parallelism

Quantum parallelism [43] is a quantum-mechanical effect we can use to build algorithms. In many applications, such as search, the problem is often given as a Boolean function f, which is solved by querying f several times. To employ a quantum algorithm to solve this problem, the first step is to encode the function f with a unitary U_f, called a quantum oracle. Given the effect of superposition, we can compute $f(x)$ for all x simultaneously by querying the oracle just once (using a superposed state containing all $|x\rangle$'s).

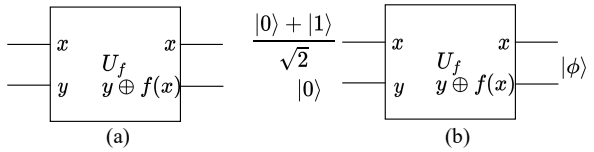

Fig. 1. Quantum parallelism

Figure 1(a) shows a typical circuit design for such a unitary. The input state is a superposition including all computational basis states x, and the results $f(x)$ are stored in the ancilla qubit. The general effect is:

$$U_f(\sum_{x=0}^{2^n-1} a_x |x\rangle |y\rangle) = \sum_{x=0}^{2^n-1} a_x |x\rangle |y \oplus f(x)\rangle$$

However, taking advantage of the benefit of quantum parallelism is often challenging. To illustrate, consider the case when setting the ancilla qubit to $|0\rangle$ in Figure 1(b). The effect of this oracle is:

$$U_f(\frac{|00\rangle + |10\rangle}{\sqrt{2}}) = \frac{|0, f(0)\rangle + |1, f(1)\rangle}{\sqrt{2}}$$

Although we have computed and stored the information of $f(x)$ for all x in the last qubit, we cannot retrieve them by simply measuring the qubits. The restriction is the same as the limitation of measurement. We can only get one result: 0, $f(0)$ or 1, $f(1)$ from one measurement because state $|\phi\rangle$ is a superposition. The superposition will collapse to a computational basis state when being measured. In general, suppose the input is an n-qubit register, then we can only get one pair of results: x, $f(x)$ when being measured, though 2^n $f(x)$ are evaluated simultaneously. Thus, we need ingenious ways to harness the power of quantum parallelism.

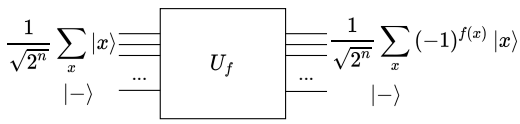

Fig. 2. Phase kickback

Phase kickback trick is a mechanism to retrieve information. The trick is to set the ancilla qubit as $|-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$, as shown in Figure 2. The phase kickback trick marks the state $|x\rangle$ satisfying $f(x) = 1$ with a minus sign, which can be further utilized in algorithms like Grover's search. The effect of this trick is:

$$U_f(\sum_{x=0}^{2^n-1} a_x |x\rangle |-\rangle) = \sum_{x=0}^{2^n-1} (-1)^{f(x)} a_x |x\rangle |-\rangle$$

2.4 Grover's Search

Grover's search is the crucial technique to improve DTC-based static analysis algorithms in this paper. It solves the problem of finding a target x satisfying $f(x) = 1$ from an unsorted search space with N elements in $O(\sqrt{N})$ time, whereas classical methods solve it in $O(N)$ time. We say that x is a target in the following illustration when $f(x) = 1$. We say x is a nontarget when $f(x) = 0$.

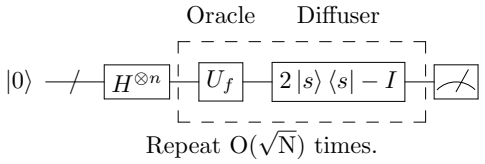

Fig. 3. Grover's search

The procedure of Grover's search has three components:

Initialization: This step aims to construct an equal-probability superposition with all computational basis states. Recall when applying a Hadamard gate on $|0\rangle$, a superposition with equal probability of $|0\rangle$ and $|1\rangle$ is generated:

$$H(|0\rangle) = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = |+\rangle$$

In general, after applying n Hadamard gates on n qubits all initialized with $|0\rangle$, the superposition consists of all computational basis states with equal probability:

$$\begin{aligned} & H(|0\rangle) \otimes H(|0\rangle) \otimes \ldots \otimes H(|0\rangle) \\ =& \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \ldots \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ =& \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \end{aligned} \tag{1}$$

Oracle: The effect of the oracle is to split data into two groups (target or nontarget) simultaneously by marking those $|x\rangle$ satisfying $f(x) = 1$ with a minus sign:

$$\sum_{x=0}^{2^n-1} a_x |x\rangle \rightarrow \sum_{y:\, f(y)=0} a_y |y\rangle - \sum_{z:\, f(z)=1} a_z |z\rangle \tag{2}$$

Diffusion: By applying $2|s\rangle\langle s| - I$ where $|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$, the amplitude (coefficient) of $|x\rangle$ with $f(x) = 1$ is increased to improve the probability of observing x:

$$\sum_{x=0}^{2^n-1} a_x |x\rangle \rightarrow \sum_{x=0}^{2^n-1} (2A - a_x) |x\rangle, \tag{3}$$

where A is the mean of the original amplitudes: $A = \frac{1}{2^n} \sum_{x=0}^{2^n-1} a_x$.

Grover's search repeats the oracle and diffusion steps $O(\sqrt{N})$ times, as in Figure 3. The number of repetitions is called *Grover iteration*, where one Grover iteration consists of one oracle call and one diffusion process. In each repetition, the amplitudes of the target states are increased to boost the probability of observing them. Finally, we get a solution of $f(x) = 1$ with a high probability by measuring the final state.

3 QUANTUM SPEEDUPS FOR DTC-BASED STATIC ANALYSIS

We first introduce a motivating example to describe the quantum speedups in searching subtasks in Section 3.1. Then, we provide an effective oracle implementation in Section 3.2, yielding the time complexity $O(\sqrt{N}polylog(N))$. We illustrate our quantum searching strategy as a subroutine for DTC-based static analysis algorithms in Section 3.3. Finally, we explain why our quantum approach produces consistent results with classical approaches in Section 3.4.

Table 1. row i and row j from an example graph with eight nodes.

	i	j	k	l	m	n	o	p
$i(v_2)$	0	1	1	0	0	0	1	0
$j(v_1)$	0	0	0	0	0	0	1	1

3.1 Motivating Example: Quantum Speedups for Searching Subtasks

Dynamic transitive closure-based static analysis algorithms typically consist of several searching subtasks, which are challenging to improve using classical approaches. To find a way to speed up DTC-based static analysis algorithms, we are inspired by quantum graph algorithms. The following gives a motivating example of the quantum speedup on a DTC-based graph reachability analysis.

In a DTC-based static analysis problem, the searching subtask is to find new reachable information based on current reachability. For illustration, we consider a dynamic transitive closure problem in a given graph G with N nodes, where $\langle i, j\rangle$ (an edge from i to j) is given, and then the searching subtask is to look for k such that $\langle j, k\rangle$ exists, but $\langle i, k\rangle$ does not. Storing G uses the adjacency matrix model (i.e., an $N \times N$ Boolean matrix A where $A[i][j] = 1$ iff $\langle i, j\rangle \in G$). The classical implementation idea is to retrieve row j as vector v_1 and row i as vector v_2 from the adjacency matrix and then to search all indices *index* such that $v_1[index] = 1$ and $v_2[index] = 0$, which takes linear time. If we can apply Grover's search to find all indices instead of an exhaustive search, we may have a quadratic speedup for each searching subtask to reduce the total running time.

Consider a DTC example for $N = 8$, where the current processing edge is $\langle i, j\rangle$, the row j (v_1) and row i (v_2) from the adjacency matrix are shown in Table 1. Classically, we need to check each of these eight candidates (corresponding to columns i to p) individually to find new reachable nodes from i. The classical evaluations of the first seven nodes return false, and the last returns true, resulting in a total of eight processing iterations. By using quantum search, the number of iterations can be reduced to two by Grover's search for this example. We use binary numbers here to represent the node indices (e.g., 000 to 111 for columns i to p) of the eight nodes.

Applying Equation 1 in Grover's search, the initial superposition can be obtained as:

$$\frac{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle}{\sqrt{8}}$$

After the oracle step, only the sign of $|111\rangle$ is changed by Equation 2.

$$\frac{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle - |111\rangle}{\sqrt{8}}$$

Using Equation 3 to compute the superposition after diffusion, the mean of all coefficients is $\frac{6}{8\sqrt{8}}$, applying $a_{new} = 2 * mean - a_{old}$, and the new superposition is:

$$\frac{1}{2\sqrt{8}}\left(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle\right) + \frac{5}{2\sqrt{8}}|111\rangle$$

Then, we need to repeat the oracle and diffusion steps. After the oracle step using Equation 2, only the sign of $|111\rangle$ is changed:

$$\frac{1}{2\sqrt{8}}\left(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle\right) - \frac{5}{2\sqrt{8}}|111\rangle$$

In the diffusion step, the mean is $\frac{1}{8\sqrt{8}}$. After updating the coefficient using Equation 3, the superposition is obtained as:

$$-\frac{1}{4\sqrt{8}}\left(|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle\right) + \frac{11}{4\sqrt{8}}|111\rangle$$

The above process takes two Grover iterations consisting of two oracle calls and two diffusion processes. The final state has a $(\frac{11}{4\sqrt{8}})^2 \approx 94\%$ chance of getting 111 (Node p) when measured, so we have a 94% chance of getting the target 111 (Node p) in only two iterations using Grover's search. Note that the probability of getting a target node can be increased by running the whole algorithm more times. For instance, running one additional time will make the probability of getting a target $94\% + 6\% \times 94\% = 99.64\%$. More generally, we can increase the probability close to 100% by running Grover's search $O(\log N)$ times [12], provided that the output solution can be efficiently verified. The detail is discussed in Section 3.3. The practical implementation of Grover's search for DTC-based static analysis needs to consider the oracle implementation to compute $v_1[i] \wedge \neg v_2[i]$, which is omitted in this motivating example. We discuss our oracle design in the following section (Section 3.2).

3.2 Oracle Implementation

The complexity $O(\sqrt{N})$ of Grover's search discussed in Section 2.4 is the query complexity, which measures the number of calls to an oracle without knowing its internal design. A quantum algorithm with smaller query complexity does not necessarily mean it will have the actual speedup because the cost of implementing the circuit for the oracle can be high. To obtain the overall time complexity, this section details the oracle implementation of our quantum search approach.

There are two challenges in implementing an efficient oracle without adversely impacting the time complexity. The first challenge is how to encode classical data into quantum superposition. Loading classical data into a quantum computer is an important and emerging topic in the quantum community. As most quantum algorithms are faster than $O(N)$, the bottleneck in I/O, which typically takes time $O(N)$, becomes a significant problem in quantum computing. The second challenge is how to implement the classical procedure in quantum computing for DTC-based analysis. If the cost of implementing the oracle is $O(N)$ with respect to the search space size N, it will incur a higher cost than $O(\sqrt{N})$ in each searching subtask; hence the overall solution is no longer superior to the classical one.

Model: We leverage QRAM to implement our oracle. QRAM loads N data into quantum superposition in $O(polylog(N))$ time [29, 44], which will make the theoretical quantum speedup stand out. Suppose there are $N = 2^n$ classical data stored in a QRAM; the QRAM LOAD operation works as follows:

$$\sum_{i=0}^{2^n-1} a_i |i\rangle |0\rangle \xrightarrow{LOAD} \sum_{i=0}^{2^n-1} a_i |i\rangle |d_i\rangle ,$$

where d_i is the data stored at location i. A few feasible implementations of QRAM have been proposed though there is no physical deployment of QRAM in quantum computers for now. For Grover's search, a binary tree structure has been proposed to tackle database applications [43], which shows the direction of solving the quantum I/O problem. A bucket brigade architecture for a QRAM [29] is designed to reduce the number of switches from linear to $O(polylog(N))$, providing exponential speedup on the addressing scheme. Recently, reading and writing querying can be done in $O(polylog(N))$ time using gate parallelism in terms of circuit-depth complexity [44]. Recursively loading classical data into a quantum machine takes $O(polylog(N))$ time in terms of circuit-depth complexity [21].

Oracle: Given the search space of size N, we need $n = \lceil \log N \rceil$ address qubits, two ancilla qubits set to $|0\rangle$, and one ancilla qubit set to $|-\rangle$. Three steps are then conducted to realize the unitary:

$$\sum_{i=0}^{2^n-1} a_i |i\rangle \rightarrow \sum_{i=0}^{2^n-1} (-1)^{v_1[i] \wedge \neg v_2[i]} a_i |i\rangle$$

for any a_i. Suppose we are provided with the QRAMs for both v_1 and v_2:

- **Step 1:** For any input state $\sum_{i=0}^{2^n-1} a_i |i\rangle$, we append ancilla qubits and use QRAM LOAD for v_1 and v_2:

$$\sum_{i=0}^{2^n-1} a_i |i\rangle |0\rangle |0\rangle |-\rangle \xrightarrow{LOAD} \sum_{i=0}^{2^n-1} a_i |i\rangle |v_1[i]\rangle |v_2[i]\rangle |-\rangle$$

- **Step 2:** We use a Boolean function $f = v_1[i] \wedge \neg v_2[i]$ to distinguish targets (solutions of $f(x) = 1$) from nontargets (solutions of $f(x) = 0$) for DTC-based static analysis. The function can be implemented by applying an X gate on the second data qubit to compute $\neg v_2[i]$, followed by a Toffoli gate to compute the result $f = v_1[i] \wedge \neg v_2[i]$ and store it in the last qubit. When setting the last qubit to $|-\rangle$, the phase kickback (discussed in Section 2.3) works. After this, we need to apply an X gate on the second data qubit to recover its state to $v_2[i]$:

$$\sum_{i=0}^{2^n-1} a_i |i\rangle |v_1[i]\rangle |v_2[i]\rangle |-\rangle \rightarrow \sum_{i=0}^{2^n-1} (-1)^{v_1[i] \wedge \neg v_2[i]} a_i |i\rangle |v_1[i]\rangle |v_2[i]\rangle |-\rangle$$

- **Step 3:** Finally, we apply the QRAM LOAD one more time to set ancilla qubits to $|0\rangle$, and the effect of the oracle is realized if the ancilla qubits are omitted:

$$\sum_{i=0}^{2^n-1} (-1)^{v_1[i] \wedge \neg v_2[i]} a_i |i\rangle |v_1[i]\rangle |v_2[i]\rangle |-\rangle \rightarrow \sum_{i=0}^{2^n-1} (-1)^{v_1[i] \wedge \neg v_2[i]} a_i |i\rangle |0\rangle |0\rangle |-\rangle$$

In short, the oracle = LOAD + X + Toffoli + X + (UN)LOAD. The cost of LOAD, and also the oracle, is $O(polylog(N))$, while the cost of the diffusion operator is $O(\log N)$ [22]. Thus, the total time complexity of our implementation of Grover's search is $O(\sqrt{N} polylog(N))$.

Note that our approach needs to mark the found target x as a nontarget (i.e., update $v_2[x] = 1$) after each successful search, which can be done efficiently without increasing the cost of the oracle. In the QRAM model, data can be stored in a classical database [43], and modifying a single item in the classical database does not impact the quantum circuit. Hence, the complexity of the oracle remains unaffected.

3.3 Quantum Searching Subroutine for DTC-Based Static Analysis

After discussing the oracle implementation, we can now use time complexity to measure the cost of our overall approach. To reduce the cost of searching subtasks in DTC-based static analysis algorithms, we use an approach that is based on a general Grover's search algorithm. This approach enables us to locate a single target ($f(x) = 1$) within a search space of size N, containing M targets, with an expected number of queries of $O(\sqrt{N/M})$, even without prior knowledge of M. It is worth noting that the number of queries corresponds to the calls made to the quantum oracle, where each Grover iteration comprises one oracle call and one diffusion process. To address the probabilistic nature of quantum computing, we employ an efficient probability amplification technique to achieve deterministic static analysis. This approach enhances the probability of obtaining accurate results with confidence, compensating for the inherent probabilistic outcomes in quantum computations.

Algorithm 1: Grover Search

Input: Quantum oracle O_f, iteration number j
Output: x which might be a target

1 Initialize superposition $|\phi\rangle$ as discussed in Section 3.2 and set *iteration* = 0
2 **for** *iteration* < j **do**
3 $|\phi\rangle = O_f |\phi\rangle$ // Apply Oracle.
4 $|\phi\rangle = U_D |\phi\rangle$ // Apply Diffusion.
5 *iteration* += 1
6 Measure according to the computational basis and return the outcome

Algorithm 2: Quantum Search for Finding One out of M Targets from N Elements

Input: Search space size N, classical $O(1)$-time verification function $VERIFY(x)$, quantum oracle O_f
Output: One target x with $VERIFY(x) = True$ or -1 meaning no target is found

1 Initialize $m = 1$, $\lambda = \frac{6}{5}$ and *total* = 0
2 j = randint(0, m) // Choose j uniformly at random among the non-negative integers smaller than m.
3 x = Algorithm1(O_f, j) // Apply j iterations of Grover's search and then observe the index register to get an output x.
4 *total* += j
5 **if** $VERIFY(x)$ == *True* **then**
6 return x // Successfully find a target x.
7 **else**
8 **if** *total* $\geq \sqrt{N}$ **then**
9 return -1 // No target with a high probability.
10 $m = \min(\lambda m, \sqrt{N})$ // Increase m for the next attempt.
11 **go to** line 2 // Try search more times.

We present Algorithm 1 as a formulation of Grover's search, allowing for a specified number of Grover iterations (j queries) as input. By executing j iterations of Grover's search, the probability of observing targets is increased, although reaching a very high probability is not guaranteed. The more general implementation of Grover's search with an unknown number of targets (i.e., M is unknown) [11] is illustrated in Algorithm 2, which finds one target in $O(\sqrt{N/M})$ expected queries. Algorithm 2 begins with a random guess ($m = 1$ at Line 1) and iteratively calls Grover's search (Line 3) by adjusting the values of m and j (Line 10 and back to Line 2) until a target index is found (Line 6), or no target is observed after $O(\sqrt{N})$ queries, indicated by the return value -1 (Line 9). Note that although returning x at Line 6 ensures that x is indeed a target, returning -1 at Line 9 does not necessarily mean no target in the current database. However, repeating the search $\sqrt{N}$ times until claiming so (Line 8) guarantees that the probability of making this type of mistake is exponentially small [11, Theorem 3].

Lemma 3.1. *[11, 23] Algorithm 2 uses Grover's search and finds one out of M targets after an expected number of at most $0.9\sqrt{N/M}$ queries (number of calls to the oracle) with a high probability.*

The output of the subroutine is uniformly at random among the M targets. The algorithm does not need prior knowledge of M before searching.

Theorem 3.2. *The time complexity of Algorithm 2 in DTC-based static analysis problems is* $O(\sqrt{N/M}polylog(N))$.

Proof. According to Lemma 3.1, the expected query complexity of Algorithm 2 is $O(\sqrt{N/M})$. Each query consists of two steps: oracle and diffusion with the cost of $O(polylog(N))$ and $O(\log N)$, respectively, yielding the total time complexity $O(\sqrt{N/M}polylog(N))$. Note that the time complexity discussed in this paper is the worst-case expected time complexity due to the probabilistic nature of Grover's search. □

In DTC-based static analysis, we need to find all M targets instead of one target. Hence we need to analyze the cost of finding all M targets based on Theorem 3.2. To eliminate the possibility of re-finding a target that was already found in the previous rounds, we need to mark the found target as a nontarget in QRAM and decrease the number of targets by one in the next search.

Theorem 3.3. *All M targets from an N-size search space can be found in* $O(\sqrt{NM}polylog(N))$ *time.*

Proof. As discussed in Section 3.2, data of QRAM can be stored in a classical database. One data can be modified efficiently in the classical database without increasing the oracle overhead. When we have found i targets, the number of targets in the search space becomes $M - i$, so the cost of finding the i-th target is $O(\sqrt{N/(M-i)}polylog(N))$. Therefore, the total cost becomes

$$
\begin{aligned}
&O(\sqrt{N/M}polylog(N)) + O(\sqrt{N/(M-1)}polylog(N)) + \cdots + O(\sqrt{N}polylog(N)) \\
=&O(\sqrt{N}polylog(N)\sum_{k=1}^{M}\frac{1}{\sqrt{k}}) \approx O(\sqrt{N}polylog(N)\int_{1}^{M+1}\frac{1}{\sqrt{x}}dx) \\
=&O(2\sqrt{N}polylog(N)(\sqrt{M+1}-1)) = O(\sqrt{NM}polylog(N)).
\end{aligned}
$$

In the end, an additional $O(\sqrt{N}polylog(N))$ time is needed to claim we have found all targets, but this does not add to the total (asymptotic) cost. □

Note that even if there are some targets, there is a very small chance that Algorithm 2 may fail to find them. To deal with this issue, we call Algorithm 2 at most $c \log N$ times where c is a small integer (e.g., ≥ 3) for each target (Line 3) in Algorithm 3, hence taming the error probability to yield consistent results as the classical approach as explained in Section 3.4. An index x is returned after each call (Line 4). If $x \neq -1$, it is a target (verified in Algorithm 2), then added to the result list and marked as a nontarget in the following rounds (Lines 5-7) per discussed in Theorem 3.3. Note that the oracle O_f is updated at this step without increasing its execution cost, as discussed in Section 3.2. When a target is found, we restore the *iteration* and continue looking for the next target (Line 8). If $x = -1$ (Algorithm 2 claims no target with a high probability), Algorithm 3 continues searching (Lines 9-10) until Algorithm 2 continuously claims no target $c \log N$ times.

We prove that our search subroutine (each time when Algorithm 3 is called) successfully finds all M targets from N elements with probability at least $(1 - \frac{1}{N^c})^M$ (Theorem 3.4). The overall success probability of applying Algorithm 3 is at least $1 - \frac{1}{N^{c-2}}$ (Theorem 3.5) when used in DTC-based static analysis to generate consistent results with the classical counterpart.

Theorem 3.4. *Algorithm 3 finds all M targets from an N-size search space in* $O(\sqrt{NM}polylog(N))$ *time with a success probability at least* $(1 - \frac{1}{N^c})^M$*, where c is a small integer* ≥ 3.

Algorithm 3: Improved Search Subroutine for Finding M Targets from N Elements

Input: Vector v_1 and v_2, search space size N, classical $O(1)$-time verification function *VERIFY*(x)

Output: List L containing all targets

1 Set *iteration* = 0 and c to a small integer (e.g., ≥ 3) // Futher explained in Section 3.4
2 The oracle O_f is implemented using the approach discussed in Section 3.2
3 **while** *iteration* $< c \log N$ **do** // Probability amplification.
4 x = Algorithm2(N, *VERIFY*, O_f) // Get candidate x in $O(\sqrt{N/M}\text{polylog}(N))$.
5 **if** $x \neq -1$ **then**
6 L.append(x) // x is indeed a target.
7 $v_2[x] = 1$ // Mark x as a nontarget in QRAM.
8 *iteration* = 0 // Begin searching the next target.
9 **else**
10 *iteration* += 1 // Algorithm 2 claims no target exists.
11 return L

Proof. Because Grover's search is a probabilistic algorithm and each subroutine has a constant error rate ($< \frac{1}{2}$), we use a classical probability amplification, which bounds the error probability to $\epsilon > 0$ by running an algorithm with a constant error rate independently $\Theta(\log(1/\epsilon))$ times [12]. Because DTC-based static analysis has $O(N^2)$ targets to find, we set $\epsilon = \frac{1}{N^c}$, which runs Algorithm 2 at most $c \log N$ times (Lines 3-10) to find one target to bound the probability of missing a target to $\frac{1}{N^c}$. The probability of successfully finding all M targets is thus at least $(1 - \frac{1}{N^c})^M$, and we are done. Given Theorem 3.3, the time complexity of Algorithm 3 is $O(\sqrt{NM}polylog(N) \times c \log N) = O(\sqrt{NM}polylog(N))$. Note that probability amplification can be used because an output x can be tested/verified in $O(1)$ time using a classical computer. □

Theorem 3.5. *The probability of applying Algorithm 3 on DTC-based static analysis to generate consistent results with the classical approach is at least* $1 - \frac{1}{N^{c-2}}$, *where* c *is a small integer* ≥ 3.

Proof. The DTC-based static analysis algorithms generally need to find $O(N^2)$ targets. According to Theorem 3.4, the success probability of finding all M targets from N-size search space in one call to Algorithm 3 is at least $(1 - \frac{1}{N^c})^M$. The DTC-based static analysis calls Algorithm 3 multiple times to find all solutions. Suppose the i-th call has M_i targets to find and $0 \leq M_1 + M_2 + \cdots \leq N^2$. The success probability of finding all solutions is $(1-\frac{1}{N^c})^{M_1} \times (1-\frac{1}{N^c})^{M_2} \times \cdots = (1-\frac{1}{N^c})^{M_1+M_2+\ldots} \geq (1-\frac{1}{N^c})^{N^2} \geq 1 - \frac{1}{N^{c-2}}$. Hence, we improve the probability of finding all solutions (consistent results) in DTC-based static analysis to at least $1 - \frac{1}{N^{c-2}}$. □

3.4 Consistent results with the classical method

The deterministic output is a fundamental expectation for most static analysis algorithms, necessitating an explanation of why quantum algorithms can be applied and accepted within the static analysis community. Our quantum approach is precision-preserving. Though Grover's search is probabilistic in nature, Algorithm 2 [11] can almost achieve the certainty of finding the searching targets. Algorithm 3 (our subroutine) further tames the probability issue to improve Algorithm 2 to find all M targets. Note that even the current classical hardware devices have a very small probability of error, called the hardware failure rate [35]. Therefore, from a practical point of view,

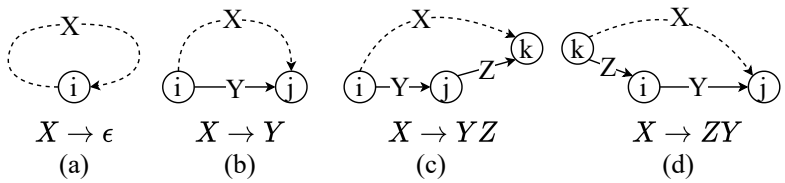

Fig. 4. Four cases of adding an edge in CFL-reachability

a probabilistic algorithm whose error probability is less than the hardware failure rate is acceptable (and can be safely regarded as a deterministic one). In this paper, we use probability amplification to reduce the error probability of our algorithm to a desirable level. Theorem 3.5 shows that our algorithm produces consistent results with classical algorithms with a probability of at least $1-\frac{1}{N^{c-2}}$, where c is a constant no less than 3. For a large N, it is safe to set c to 3 because $\frac{1}{N}$ is close to 0 if N is a very large number. For a smaller N, we can adjust c to reduce the error probability to small enough. As an example, if the input size is $N = 10^4$, we can set $c = 4$ to have a bounded error probability 10^{-8}.

4 APPLICATION I: QUANTUM SPEEDUPS ON CFL-REACHABILITY

In this section, we introduce the quantum speedups on CFL-reachability analysis and then move to set constraints solved in Section 5. CFL-reachability is a typical algorithm used in multiple static analysis techniques, like interprocedural slicing, data-flow analysis, shape analysis, etc. In this section, we first formulate the conventional CFL-reachability algorithm and analyze its complexity in Section 4.1. After this, we show how to improve it with our search subroutine and analyze the improved complexity in Section 4.2. Finally, we provide a running example to illustrate the improvement brought by our quantum solution in Section 4.3.

4.1 Classical CFL-Reachability Algorithm

Definition. Let $CFG = (\Sigma, \mathbb{N}, \mathbb{P}, \mathbb{S})$ be the context-free grammar (alphabet Σ, nonterminal symbols $\mathbb{N}$, production rule $\mathbb{P}$, and start symbol $\mathbb{S}$). Given a CFG and an edge-labeled graph $G = (V, E)$, where $A\ \langle i, j\rangle \in G$ means a directed edge from i to j with a terminal or nonterminal A. An S-path is a sequence of edge labels following the path order in G. We denote two nodes (Nodes *src* and *snk*) as S-reachable if there is an S-path from *src* to *snk*.

The CFL-reachability problem has four variants: single-source, single-target, single-source-single-target, and all-pairs S-path problems. We consider the all-pairs S-path problem to find all S-reachable nodes for each node because this problem is the most complex one and also suffers the cubic bottleneck. Other three problems can be considered as a subset of the all-pairs S-path problem and can also benefit from the improvement made to the all-pairs S-path problem.

Algorithm. The pseudocode of CFL-reachability is given in Algorithm 4. The all-pairs CFL-reachability computes the transitive closure of a graph based on the production rules shown in Figure 4:

a) *Lines 2-5:* All nodes have a self-pointed edge labeled with X if there is a production rule $X \to \epsilon$.
b) *Lines 8-10:* Add an edge $X\ \langle i, j\rangle$ if there is an edge $Y\ \langle i, j\rangle$ and a production rule $X \to Y$.
c) *Lines 11-14:* Add an edge $X\ \langle i, k\rangle$ if there are edges $Y\ \langle i, j\rangle$, $Z\ \langle j, k\rangle$, and a production rule $X \to YZ$.

Algorithm 4: CFL-Reachability Baseline Algorithm

Input: Edge-labeled directed graph $G = (V, E)$, normalized $CFG = (\Sigma, N, P, S)$
Output: Set $\{(i, j) \mid S\ \langle i, j\rangle \in G\}$

```
1  Initialize worklist W and add E to W
2  foreach production X → ϵ ∈ P do
3      foreach node v ∈ V do
4          if X ⟨v, v⟩ ∉ E then
5              add X ⟨v, v⟩ to E and to W
6  while W is not empty do
7      Select and remove an edge Y ⟨i, j⟩ from W
8      foreach production X → Y ∈ P do
9          if X ⟨i, j⟩ ∉ E then
10             add X ⟨i, j⟩ to E and to W
11     foreach production X → YZ ∈ P do
12         foreach outgoing edge Z ⟨j, k⟩ from node j do
13             if X ⟨i, k⟩ ∉ E then
14                 add X ⟨i, k⟩ to E and to W
15     foreach production X → ZY ∈ P do
16         foreach incoming edge Z ⟨k, i⟩ to node i do
17             if X ⟨k, j⟩ ∉ E then
18                 add X ⟨k, j⟩ to E and to W
```

d) *Lines 15-18:* Add an edge $X\ \langle k, j\rangle$ if there are edges $Z\ \langle k, i\rangle$, $Y\ \langle i, j\rangle$, and a production rule $X \rightarrow ZY$.

Theorem 4.1. *Let $|\Sigma|$ be the size of the elements of terminals and nonterminals in the normalized grammar, and N be the number of nodes in the graph; then the total cost is $O(|\Sigma|^3 N^3)$.*

Proof. There may be $O(|\Sigma|N)$ outgoing and incoming edges for each node, so the total number of edges is bounded by $O(|\Sigma|N^2)$. The cost of Lines 1-5 is trivial because the cost of Line 1 is $O(|\Sigma|N^2)$, and the cost of Lines 2-5 is $O(|\Sigma|N)$. The worklist W may store $O(|\Sigma|N^2)$ edges, so the while loop in Line 6 may repeat $O(|\Sigma|N^2)$ times. The total cost of Lines 8-10 is $O(|\Sigma|N^2) \times O(|\Sigma|) = O(|\Sigma|^2 N^2)$. Lines 11-14 and Lines 15-18 may repeat $O(|\Sigma|^2 N)$ leading to the total cost of Lines 11-14 and Lines 15-18 being $O(|\Sigma|N^2) \times O(|\Sigma|^2 N) = O(|\Sigma|^3 N^3)$. In the CFL-reachability problem, we can treat $|\Sigma|$ as a constant factor to the number of nodes N, so the total cost is bounded by $O(N^3)$. □

4.2 Quantum CFL-Reachability Algorithm

To reduce the complexity of the traditional CFL-Reachability algorithm (Algorithm 4), we focus on improving its searching subtasks (Lines 11-18). In this case, we can use the searching subroutine (i.e., Algorithm 3 in Section 3.3) to speed up the key searching subtasks. Note that we use the matrix model (i.e., an $N \times N \times |\Sigma|$ Boolean matrix A where $A[i][j][Y] = 1$ iff $Y\ \langle i, j\rangle \in G$) to represent the graph G with N nodes and $|\Sigma|$ alphabets, where $G[i][:][X]$ represents the i-th row with label X, and $G[:][i][X]$ represents the i-th column with label X.

Hence, Lines 11-14 are replaced by:

foreach *production* $X \rightarrow YZ \in P$ **do**
  $v_1 = G[j][:][Z],\ v_2 = G[i][:][X]$
  $L = Algorithm3(v_1, v_2, N, lambda\ x : v_1[x] = 1\ and\ v_2[x] = 0)$
  foreach k *in* L **do**
    add $X\ \langle i, k\rangle$ to E and to W

Similarly, Lines 15-18 are replaced by:

foreach *production* $X \rightarrow ZY \in P$ **do**
  $v_1 = G[:][i][Z],\ v_2 = G[:][j][X]$
  $L = Algorithm3(v_1, v_2, N, lambda\ x : v_1[x] = 1\ and\ v_2[x] = 0)$
  foreach k *in* L **do**
    add $X\ \langle k, j\rangle$ to E and to W

Theorem 4.2. *The time complexity of the CFL-reachability algorithm using our quantum search subroutine is* $O(|\Sigma|^3 N^2 \sqrt{N} polylog(N))$.

Proof. The cost of Lines 1-5 and the total cost of Lines 7-10 remain unchanged. The cost of Lines 11-14 and Lines 15-18 are reduced. The cost of finding new edges of Lines 12-14 is reduced from $O(N)$ to $O(\sqrt{Nt} polylog(N))$ according to Theorem 3.4, where t ($\leq N$) is the number of targets. Note that for loop cost below the Algorithm 3 is $O(t)$, which is less than $O(\sqrt{Nt})$. Thus, the total cost of Lines 11-14 becomes:

$$
\begin{aligned}
&\sum_{w \in W} \sum_{X \rightarrow YZ} \sqrt{Nt} polylog(N) \\
\leq &|\Sigma|^2 polylog(N) \sum_{w \in W} \sqrt{Nt} \\
\leq &|\Sigma|^2 polylog(N) \sqrt{\sum_{w \in W} N} \sqrt{\sum_{w \in W} t} \\
\leq &|\Sigma|^2 polylog(N) \sqrt{N \times |\Sigma| N^2} \sqrt{|\Sigma| N^2} \\
= &O(|\Sigma|^3 N^2 \sqrt{N} polylog(N))
\end{aligned}
$$

The first inequality holds because the number of possible $X \rightarrow YZ$ is bounded by $O(|\Sigma|^2)$ for a fixed Y. The second inequality is the Cauchy-Schwarz inequality [58]. For the third inequality, the number of edges is bounded by $O(|\Sigma|N^2)$, and the number of elements $w \in W$ is also bounded by $O(|\Sigma|N^2)$. Thus, $\sqrt{\sum_{w \in W} N}$ is bounded by $O(\sqrt{N \times |\Sigma| N^2})$. Because the algorithm adds $X\ \langle i, k\rangle$ to W when finding a $X\ \langle i, k\rangle$, the term $\sqrt{\sum_{w \in W} t}$ is the same as the number of elements that may appear in worklist W, which is $O(\sqrt{|\Sigma| N^2})$. Finally, the upper bound of this algorithm is $O(|\Sigma|^3 N^2 \sqrt{N} polylog(N))$. Because $|\Sigma|$ is a constant factor to N, the algorithm is bounded by $O(N^2 \sqrt{N} polylog(N))$. The proof of Lines 15-18 is the same as that of Lines 11-14. Finally, the algorithm outputs the same results as the classical approach based on Theorem 3.5. □

4.3 Quantum Speedups on a CFL-Reachability Example

Figure 5 gives an eight-node graph example to illustrate the quantum speedups on CFL-reachability. We aim to show the differences between our approach and the classical method in finding one or multiple targets on the graph. Note that we omit the logarithmic factor due to a small searching space (i.e., set the number of iterations to 1 instead of $c \log N$ at Line 3 of Algorithm 3). In the example, we compute the number of quantum iterations and the corresponding success probability

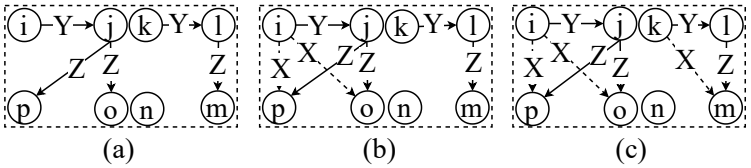

Fig. 5. Example input graph.

at each step. We use the matrix model (i.e., an $N \times N \times |\Sigma|$ Boolean matrix A where $A[i][j][Y] = 1$ iff $Y\langle i, j\rangle \in G$), which has a fixed searching dimension, to represent the graph G with N nodes and $|\Sigma|$ alphabets in the following analysis; therefore, N iterations are required to check all the neighbors of a given node.

The initial graph shown in Figure 5 (a) contains only one production rule: $X \rightarrow YZ$. Accordingly, the worklist W is initialized with five labeled edges: $\{Y\langle i, j\rangle, Z\langle j, p\rangle, Z\langle j, o\rangle, Y\langle k, l\rangle, Z\langle l, m\rangle\}$. There is no edge added in Line 2 because our example does not contain the production $X \rightarrow \epsilon$. After this, the algorithm enters the while loop (Line 6). The resulting graph and worklist of each loop iteration for classical computing are as follows:

- **Loop Iteration 1:** Edge $Y\langle i, j\rangle$ is firstly popped from the worklist W. There is no execution for Lines 8-10. Lines 11-14 find new reachable edges because of a production $X \rightarrow YZ$. To do this, Lines 13-14 are repeated eight times based on the matrix model and determine that $X\langle i, p\rangle$ and $X\langle i, o\rangle$ should be added to the graph and the worklist. An additional eight iterations are needed for Lines 17-18 to check incoming edges, but no new edges are found. Then the graph is changed to Figure 5 (b), and $W = \{Z\langle j, p\rangle, Z\langle j, o\rangle, Y\langle k, l\rangle, Z\langle l, m\rangle, X\langle i, p\rangle, X\langle i, o\rangle\}$.
- **Loop Iterations 2-5:** Next, $Z\langle j, p\rangle$ is selected from W. Intuitively, the graph is unchanged, but there are 16 iterations during processing (8 iterations for checking incoming edges and 8 for checking outgoing edges). Similar processes occur when processing $Z\langle j, o\rangle$, $X\langle i, p\rangle$, and $X\langle i, o\rangle$. Each process needs 16 iterations, resulting in a total iteration of 80, and $W = \{Y\langle k, l\rangle, Z\langle l, m\rangle\}$.
- **Loop Iteration 6:** Then, $Y\langle k, l\rangle$ is selected. Searching for outgoing edges needs eight iterations, and searching for incoming edges needs additional eight iterations. A new edge $X\langle k, m\rangle$ is found and added to the graph, resulting in the graph changing to Figure 5 (c) and $W = \{Z\langle l, m\rangle, X\langle k, m\rangle\}$.
- **Loop Iterations 7-8:** Finally, $Z\langle l, m\rangle$ and $X\langle k, m\rangle$ are processed one by one. The graph reaches a fixed point because no edge is added, but the total number of iterations is increased by 32 up to 128.

When using our search subroutine, the number of iterations in searching tasks is significantly reduced. Next, we study the number of Grover iterations (one Grover iteration consists of one oracle call and one diffusion process) and the probability of getting targets. For the 14 cases in which no target is found in the example, the classical number of iterations is eight in each process. In quantum search, this is solved by approximate timeout. We set the timeout threshold as $\sqrt{N}$ (Line 8 of Algorithm 2). For this example, the timeout threshold is $\sqrt{N} \approx 3$. If the number of iterations exceeds the threshold, we stop and believe there is no more target. When processing edge $Y\langle i, j\rangle$ in Loop Iteration 1, two edges are added to the graph, which takes eight iterations. In contrast, the number of iterations is at most 8 in Algorithm 3. The subroutine Algorithm 3 first calls Algorithm 2 to find targets with m initialized to 1. The superposition modification (omit other qubits except for the index register) is shown below:

We begin with a superposition with an equal probability according to Equation 1:

$$\frac{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle + |110\rangle + |111\rangle}{\sqrt{8}}$$

Applying Equation 2, 110 (o) and 111 (p) are marked by the oracle:

$$\frac{|000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle - |110\rangle - |111\rangle}{\sqrt{8}}$$

The mean of the above superposition is $\frac{4}{8\sqrt{8}}$, which means the next superposition using Equation 3 is:

$$0 \times (\ |000\rangle + |001\rangle + |010\rangle + |011\rangle + |100\rangle + |101\rangle) + \frac{2}{\sqrt{8}} \times (|111\rangle + |110\rangle)$$

The probability of getting a target is $(\frac{2}{\sqrt{8}})^2 + (\frac{2}{\sqrt{8}})^2 = 100\%$, and we can get either 110 (o) or 111 (p). Suppose we get 110 (o) in this step; the next search is to find 111 (p). The superposition modification is the same as Section 3.1. We have about a 94% chance of finding target 111 (p). Note that we also need to deal with the case with no target. In this example, we use 1 (finding either 110 (o) or 111 (p)) + 2 (finding rest) + 3 (no more target) = 6 as the number of iterations using our approach to deal with outgoing edges from j based on edge $Y\ \langle i, j\rangle$ in Loop Iteration 1. Note that if we measure the register qubits and unfortunately fail to get the target 111 (p), the subroutine in Algorithm 3 handles this by doing more repetition, and the following search still has a 94% chance to output target 111 (p). The probability of getting a target from these two searches is improved to $94\% + 6\% \times (94\%) = 99.64\%$.

When processing outgoing edges from l based on edge $Y\ \langle k, l\rangle$ in Loop Iteration 6, one edge is added, but eight iterations are needed. In the quantum version, the superposition modification is like the one in Section 3.1, which has a 94% chance of getting a target, so the number of iterations here is reduced to 5 in the quantum version. Running the search one more time would increase the chance to 99.64%, like the previous analysis.

To sum up, the total number of iterations using our approach is $3 \times 14 + 6 + 5 = 53$ instead of $8 \times 16 = 128$ for the classical method. In the example above in this section, we give a concrete example of using our quantum subroutine to reduce the number of iterations with $N = 8$. We show that case $M = 0$ with approximate timeout and $M = 1$ with more attempts. The case $M = 2$ is a particular case when $M = \frac{N}{4}$ and the probability of finding a target is a certainty.

5 APPLICATION II: QUANTUM SPEEDUPS ON SET CONSTRAINTS

Set constraints have also been used in many static program analysis applications, including type inference, inclusion-based analysis, etc. We first detail the original set constraints reduction algorithm (called SC-Reduction) and its complexity in Section 5.1. Then, we present how to improve the algorithm using our search subroutine and analyze its improved complexity in Section 5.2. We also give an example to illustrate our improvement in Section 5.3.

5.1 Classical SC-Reduction Algorithm

Definition. A set constraint is a relation of form $V \supseteq sexp$, where V and $sexp$ are set expressions. Set expressions consist of set variables (denoted by X, Y, ...), atomic expression $c(V_1, ..., V_r)$, and projection pattern $c_i^{-1}(V)$. c is called a constructor in an atomic expression, and $c_i^{-1}(V)$ asks for the i-th element in V. In the set constraints problem, a ground term of constructors is an empty constructor or $c(V_1, ..., V_r)$ if all set variables $V_1, ..., V_r$ are ground terms.

Algorithm 5: SC-Reduction Baseline Algorithm

Input: A collection of set constraints C
Output: A collection of solved set constraints C

```
1  Initialize worklist W to {X ⊇ a ∈ C | a is a nullary constructor}
2  Mark all set variables as having the property "not ground"
3  while W is not empty do
4      Select and remove a constraint X ⊇ sexp from W
5      if X ⊇ sexp is of the form X ⊇ c(V_1, V_2, ..., V_r) then
6          foreach constraint Y ⊇ c_i^{-1}(X) in C do
7              if Y ⊇ V_i is not in C then
8                  Insert Y ⊇ V_i into C and W
9          foreach constraint Y ⊇ X in C do
10             if Y ⊇ c(V_1, V_2, ..., V_r) is not in C then
11                 Insert Y ⊇ c(V_1, V_2, ..., V_r) into C and W
12     else if X ⊇ sexp is of the form X ⊇ Y then
13         foreach constraint Y ⊇ c(V_1, V_2, ..., V_r) in C do
14             if V_1, ...V_r are all ground then
15                 if X ⊇ c(V_1, V_2, ..., V_r) is not in C then
16                     Insert X ⊇ c(V_1, V_2, ..., V_r) into C and W
17     if X is not marked as ground then
18         Mark X as ground
19         foreach constraint Y ⊇ c(...X...) in the original collection of constraints do
20             if all set variables used in c(...X...) are ground then
21                 Insert Y ⊇ c(...X...) into W
22         foreach constraint Y ⊇ X in the original collection of constraints do
23             Insert Y ⊇ X into W
```

Algorithm. The algorithm for set constraints is given in Algorithm 5. Let C be a collection of set constraints, and the constraints can be solved by repeating two steps:

a) Add constraint $X \supseteq V_i$ to C if both $X \supseteq c_i^{-1}(Y)$ and $Y \supseteq c(V_1, ...V_r)$ exist in C and $c(V_1, ..., V_r)$ is ground.
b) Add constraint $X \supseteq c(V_1, ..., V_r)$ to C if both $X \supseteq Y$ and $Y \supseteq c(V_1, ..., V_r)$ exist in C and $c(V_1, ..., V_r)$ is ground.

Theorem 5.1. *Let k be the number of atomic expressions used in C, v be the number of set variables used in C, p be the maximum number of projection constraints that can match with a given constraint of the form $Y \supseteq c(V_1, V_2, ..., V_r)$, and N be the total number of constraints in the original problem; then the time complexity of this algorithm is $O(pvk + kv^2 + N) = O(N^3)$.*

Proof. The number of constraints in format $X \supseteq c(V_1, V_2, ..., V_r)$ is bounded by $O(kv)$, and the number of constraints in format $X \supseteq Y$ is bounded by $O(v^2)$, so the while loop of Line 3 may repeat $O(kv + v^2)$ times. The for loop of Line 6 may repeat $O(p)$ times, and the loop of Line 9 may repeat

$O(v)$ times, so the total time complexity of Lines 5-11 is bounded by $O(kv \times (p+v))$. The for loop of Line 13 may repeat $O(k)$ times, and the cost of Line 14 can be ignored because r can be seen as a constant factor. Thus, the total cost of Lines 12-16 is $O(v^2 \times k)$. The checking cost in Line 17 is bounded by $O(kv+v^2)$. Although there are for loops in Lines 18-23, we can treat them as a constant factor $O(r)$ and $O(1)$ for Line 19 and Line 22, respectively. Since r is constant, the total number of propagating ground information in Lines 17-23 is bounded by $O(N)$. Thus, the total cost of this algorithm is $O(kvp + kv^2 + N)$. Because k, v, p are proportional to N in the worst case, the total cost can also be written as $O(N^3)$. □

5.2 Quantum SC-Reduction Algorithm

To improve the performance, the key is to reduce the searching cost of Lines 5-16. The method is to employ our quantum search subroutine (Algorithm 3) to replace the exhaustive search. In this case, Lines 6-8, 9-11, and 13-16 are replaced by applying Algorithm 3. To illustrate, we first consider all data are stored in the array model: *cons_sc* (*cons_sc*$[X][V_1...V_r]$ represents $X \supseteq c(V_1, ...V_r)$), *proj_sc* (*proj_sc*$[X][Y][i]$ represents $X \supseteq c_i^{-1}(Y)$) and *var_sc* (*var_sc*$[X][Y]$ represents $X \supseteq Y$).

Concretely, Lines 6-8 are replaced by:

```
foreach V_i ∈ c(V_1, V_2, ..., V_r) do
  v_1 = proj_sc[:][X][i], v_2 = var_sc[:][V_i]
  L = Algorithm3(v_1, v_2, N, lambda x : v_1[x] = 1 and v_2[x] = 0)
  foreach Y in L do
    Insert Y ⊇ V_i into C and W
```

Lines 9-11 are replaced by:

```
v_1 = var_sc[:][X], v_2 = cons_sc[:][c(V_1, ..., V_r)]
L = Algorithm3(v_1, v_2, N, lambda x : v_1[x] = 1 and v_2[x] = 0)
  foreach Y in L do
    Insert Y ⊇ c(V_1, V_2, ..., V_r) into C and W
```

Lines 13-16 are replaced by:

```
v_1 = cons_sc[Y][:], v_2 = cons_sc[X][:]
L = Algorithm3(v_1, v_2, N, lambda x : v_1[x] = 1 and v_2[x] = 0)
  foreach c(V_1, V_2, ..., V_r) in L do
    if V_1, ...V_r are all ground then
      Insert X ⊇ c(V_1, V_2, ..., V_r) into C and W
```

Theorem 5.2. *The time complexity of the SC-Reduction algorithm using the search subroutine based on Grover's search is bounded by* $O(N^2\sqrt{N}polylog(N))$.

Proof. The proof is similar to that in Theorem 4.2. Let t_i be the number of targets found if the search space has i elements. The total cost is the cost of Lines 5-11 (denoted as W_{if}) plus the cost of Lines 12-16 (denoted as W_{else}). Thus, the cost can be written as follows:

$$
\begin{aligned}
&\sum_{w \in W_{if}} cost_{Lines5-11} + \sum_{w \in W_{else}} cost_{Lines12-16} \\
=&\sum_{w \in W_{if}} (\sqrt{pt_p}polylog(N) + \sqrt{vt_v}polylog(N)) + \sum_{w \in W_{else}} \sqrt{kt_k}polylog(N) \\
\leq& polylog(N)(\sqrt{\sum_{w \in W_{if}} p}\sqrt{\sum_{w \in W_{if}} t_p} + \sqrt{\sum_{w \in W_{if}} v}\sqrt{\sum_{w \in W_{if}} t_v} + \sqrt{\sum_{w \in W_{else}} k}\sqrt{\sum_{w \in W_{else}} t_k}) \\
\leq& polylog(N)(\sqrt{kvp}\sqrt{v^2} + \sqrt{kv^2}\sqrt{vk} + \sqrt{kv^2}\sqrt{vk})
\end{aligned}
$$

$$=O(N^2\sqrt{N}polylog(N))$$

The complexity of this algorithm is calculated based on Cauchy-Schwarz inequality. Because the number of constraints in format $X \supseteq c(V_1, V_2, \ldots, V_r)$ is bounded $O(kv)$, the term $\sum_{x \in W_{if}} p$ and the term $\sum_{x \in W_{if}} v$ are bounded by pkv and kv^2. Similarly, the number of constraints in format $X \supseteq Y$ is $O(v^2)$, and the term $\sum_{w \in W_{else}} k$ is bounded by kv^2. After finding an element, the algorithm inserts a new constraint into C, so there is a relationship between the sum of t_i. Because the number of constraints in format $Y \supseteq V_i$ is $O(v^2)$, the term $\sum_{w \in W_{if}} t_p$ is $O(v^2)$. Similarly, because the number of constraints in format $Y \supseteq c(V_1, V_2, \ldots, V_r)$ is $O(kv)$, the term $\sum_{w \in W_{if}} t_v$ and $\sum_{w \in W_{else}} t_k$ are both $O(vk)$. The cost of the other lines remains unchanged and is trivial to the total cost. Since p, k, v are proportional to N, the algorithm is bounded by $O(N^2\sqrt{N}polylog(N))$. Finally, the algorithm generates consistent results with the classical approach based on Theorem 3.5. □

5.3 Quantum Speedups on a Set Constraints Example

Similar to Section 4.3, we give an example to show the improvement of our approach to its classical counterparts. Suppose we have a constraint with only four variables (U, X, Y, and Z), one constructor c, and the constraints we have are $X \supseteq Y$, $Z \supseteq c(X)$, $X \supseteq c$, and $U \supseteq c_1^{-1}(Z)$. This covers all three kinds of constraints in set constraints-based analysis.

The model we use for analysis is the array model, which supports constant access. As we have four variables and one constructor, the array is 4×4 for three kinds of constraints. Given one variable, iterating all related constraints takes four iterations using the array model.

First, we show the process in the classical computing method. The worklist W is initialized with $X \supseteq c$ according to Line 1. Then, Line 2 marks all variables with the "not ground" property. Next, the program executes into the while loop, and we show each loop iteration as follows.

- **Loop Iteration 1:** $X \supseteq c$ is selected. $X \supseteq c$ satisfies the condition in Line 5, so Lines 6-11 execute. This process takes eight iterations, but there is no modification to C and W. The program moves to Line 17 and marks X as ground. After iterating Lines 19-21, $Z \supseteq c(X)$ is inserted into W. Finally, Lines 22-23 do nothing to W. Note that the cost of Lines 17-23 is not counted in our analysis because the classical and improved quantum algorithms share the same code.
- **Loop Iteration 2:** Then, $Z \supseteq c(X)$ is selected, which meets the condition of Line 5. $W \supseteq X$ is found and added into W and C, and this searching process needs eight iterations. Z is marked as ground. The total iteration is increased by eight, being 16 after this loop iteration.
- **Loop Iteration 3:** $U \supseteq X$ is selected, which meets the condition in Line 12, but no new constraint is found during the four iterations. U is then marked as ground, and no modification is made to W. Because W is empty, the program stops, and the total number we count for analysis is $8 + 8 + 4 = 20$.

In the quantum version, the analysis is the same as in Section 4.3. For case, $M = 0$ (M is the number of searching targets), the number of iterations is the approximate timeout threshold, i.e., $\sqrt{4} = 2$ for this example. For case $M = 1$ in this example, $M = \frac{N}{4}$ means we have a 100% chance of finding the target using Grover's search. Thus, we need 1 + 2 (determine no more targets) = 3 iterations. Specifically, in Loop Iteration 1, the cost is reduced to 1 + 2 + 2 = 5. The cost of Loop Iteration 2 is reduced to 5 as well. The cost of Loop Iteration 3 is accordingly reduced to 2. Thus, the total number of iterations using our approach is 12, compared to 20 using the classical approach. This example has a small constraint size due to the page limitation, leading to a slight difference. When using constraints of a larger size, the result will appear better.

6 EXPERIMENT EVALUATION

Though our work focuses on theoretical implications and complexity analysis of DTC-based static analysis through the lens of quantum search, this section also provides some additional empirical evaluation to facilitate the understanding of our theoretical results. This includes (1) the simulation of our quantum algorithms on IBM Qiskit using randomly generated graphs and constraints for correctness validation and running time estimation, and (2) the estimation of our performance improvement using real-world benchmarks given the two aforementioned applications (i.e., CFL-reachability analysis and SC-reduction).

Implementation. To compare the performance, we implement the baseline algorithms for CFL-reachability (Algorithm 4) and SC-reduction (Algorithm 5) with the application to alias analysis. We use an LLVM-based static analysis tool, SVF [54], to generate edge-labeled graphs for CFL-reachability, and the productions are based on alias analysis [65]. The set constraints are generated from CFL-reachability using the transformation approach [42].

On the quantum side, quantum algorithms for CFL-reachability and SC-reduction are implemented by replacing the corresponding lines with quantum subroutines (Sections 4.2 and 5.2) where Grover's search is simulated by the IBM Qiskit [4] library (in Python) on the `qasm_simulator` backend. Similar to existing quantum computing research, we evaluate our approach based on Qiskit simulation instead of a real quantum machine because of hardware limitations. Most of the quantum algorithms (ours included) are based on the assumption of large-scale and noise-free quantum machines, which are unlikely to be available in the foreseeable future. The classical simulation generates the same result as a quantum machine, which can be utilized for our empirical study.

However, because classically simulating a quantum algorithm (e.g., Grover's search) is time-consuming and requires substantial resources [67], we evaluate efficiency and correctness separately on different collections of datasets using two different strategies. For correctness, we run our algorithm on the Qiskit simulator, which can produce the same results as a (noise-free) quantum machine. However, classically simulating quantum algorithms typically result in exponential slowdowns. We use small-scale and randomly generated graphs/constraints for our evaluation. For efficiency, it is clear that we cannot demonstrate the performance of quantum algorithms in classical simulators. To deal with this problem, we take real-world programs but "statically compute" (or estimate) the number of iterations used in our quantum algorithm and compare it with the number of iterations in classical algorithms. For completeness, we also simulate our algorithm on randomly generated datasets and record the number of quantum iterations required to produce 100% correct results.

Correctness Evaluation. We randomly generate the edge-labeled graphs and set constraints using Numpy [32], which is an efficient array library. Since the input data for these two problems (CFL-reachability and SC-reduction) are usually sparse, we set the probability of generating an edge/constraint between any two nodes/constraints as 20% to simulate the CFL-reachability/SC-reduction inputs. Due to the constraints on computational capability in simulating quantum algorithms, the maximum qubit number in our evaluation is $n = 8$ (on a Macbook Pro 15" with 16GB RAM and 2.6GHz 6-core Intel Core i7 processor) for simulating Grover's search, which means only input data with $N \leq 256$ nodes/constraints are considered. The experiment divides the data into six categories, each corresponding to a different number $n \in [3, 8]$ of qubits. We randomly generate twenty inputs for each category, simulate our quantum approach, and then compare the results with the classical counterpart for our correctness evaluation. The simulation results show that both clients by our quantum search approach yield the same results (100% precision and recall) as those produced by the classical method, showing that our approach is precision-preserving.

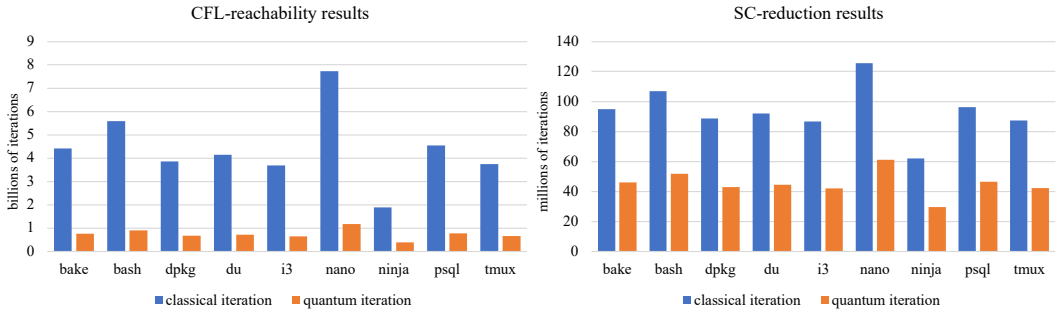

Fig. 7. CFL-reachability (LHS) and SC-reduction (RHS) estimation results comparing the numbers of classical and quantum iterations in billions and millions, respectively

.

It is interesting to mention that we observe inconsistency when the probability amplification is removed (i.e., set the number of iterations to 1 instead of $c \log N$ at Line 3 of Algorithm 3), which illustrates the necessity of probability amplification employed in Algorithm 3.

Efficiency Evaluation. Based on our simulations of randomly generated datasets, we record the number of Grover iterations required to produce 100% correct results and estimate the running time to get the trend of the running time with different input sizes. Note that we cannot use the running time of a classical simulator to estimate the time required for a quantum computer to execute the Grover search subroutine because otherwise, the Grover search would be simulated classically with similar time complexity. To address this problem, in our simulation, we replace the actual running time of simulating each Grover iteration with the theoretical estimate $\log N$. Our goal is to get the trend of running time for different input sizes N. The estimated running time is shown in Figure 6, which demonstrates that the curve of our running times (blue line) under different input sizes well aligns with the trend of complexity $O(N^{2.5} \log N)$ (orange line).

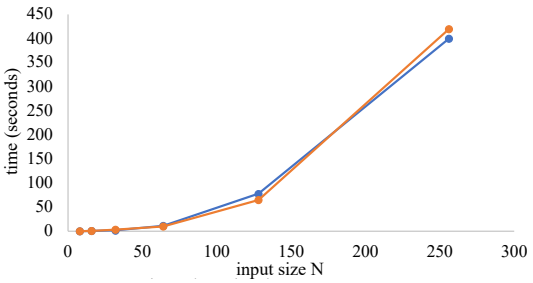

Fig. 6. The estimated running times (blue line) calculated based on Grover iterations from the simulation under different input sizes. The baseline (orange one) is the curve for $0.5 \times 10^{-4} N^{2.5} \log N$.

For the sake of completeness, we also use nine open-source programs (shown at the bottom of Figure 7) but "statically compute" the number of iterations used in our quantum algorithm and compare it with the number of iterations in classical algorithms. It is hard to compare the running times between classical and quantum algorithms due to the unavailability of practical quantum computers. Our static estimation approach is valuable because the cost per iteration is similar in the classical and quantum cases due to our efficient oracle design in Section 3.2 and the nature of the problems. The cost of one quantum (Grover) iteration is $O(polylog(N))$, because the cost of implementing the oracle is $O(polylog(N))$ as discussed in Section 3.2, and a feasible implementation of the diffusion needs $2 \log N$ Hadamard gates, $2 \log N$ X gates and $2 \log N$ Toffoli gates. In our evaluation, as other lines of Algorithms 4 and 5 remain unchanged, we only compare the number of iterations from the modified lines (Lines 11-14 and 15-18 of Algorithm 4 and Lines 6-8, 9-11 and

13-16 of Algorithm 5). The number of classical iterations is computed from the number of iterations of the loop at the corresponding lines, and the number of quantum (Grover) iterations is computed by $\lfloor 0.9\sqrt{N/M} \rfloor$ for an N-size search space with M targets based on Lemma 3.1. Figure 7 shows the estimation results of CFL-reachability and SC-reduction. The program names are shown at the bottom of the figure, and the sizes of inputs are about 30,000 nodes for CFL-reachability and 50,000 constraints for SC-reduction. The numbers of classical and quantum iterations we obtained are marked by orange and blue bars, respectively. The reduction rate is about 77% in CFL-reachability and 51% in SC-reduction.

Data Availability. The implementation and dataset are publicly available at [1].

7 DISCUSSIONS

This paper proposes the first truly subcubic solution for DTC-based static analysis by leveraging the quantum advantage of Grover's search. Grover's search employs an oracle typically defined by a Boolean function without knowledge of its design. The objective of this paper is to make necessary components explicit when utilizing Grover's search in DTC-based static analysis. These components include the oracle design (Section 3.2), the quantum search subroutine (Section 3.3), the deterministic expectation of static analysis algorithms (Section 3.4), and finally, how to apply them (Sections 4 and 5).

Limitations. However, the real-world impact of Grover's search hinges on two general assumptions prevalent in the quantum community. The first assumption pertains to the existence of large-scale and noise-free quantum machines. Currently, quantum computing operates in the noisy intermediate-scale quantum (NISQ) era [46]. The effects of noise on certain quantum algorithms, including Grover's search, are noteworthy, given the challenges associated with controlling hardware noise in current NISQ machines [47]. The second assumption involves the utilization of the well-known QRAM model for loading classical data into qubits. QRAM serves as a crucial infrastructure for solving classical problems that require the conversion of classical data into quantum states (e.g., [6]). QRAM also serves as a fundamental tool for advanced quantum data structures (e.g., quantum sets [63]). While several papers have proposed feasible implementations of QRAM (e.g., [21, 29, 43, 44]), no physical deployment currently exists. QRAM has demonstrated efficiency in terms of circuit-depth complexity [44], but the deployment of fault-tolerant QRAM remains a challenging task [41]. Although these two assumptions are independent of this paper from an algorithmic perspective, we restate them here to avoid any potential misinterpretation, with our primary focus being on the theoretical time complexity breakthrough in the cubic bottleneck of DTC-based static analysis today.

Future works. This paper improves the time complexity of DTC-based static analysis by replacing the classical exhaustive search with Grover's search without altering the structure of the algorithms. This decision is influenced by the difficulty of finding a subcubic procedure for DTC-based static analysis, as highlighted in [34]. However, it's important to note that the conclusion in [34] was drawn in the classical domain without considering the power of quantum computing. Several papers demonstrate the potential for further improving the complexity of classical problems using Grover's search by modifying algorithm structures (e.g., algorithms in [5]). This can be considered as a potential future direction for enhancing the time complexity of program analysis problems.

Moreover, we note from [52] that when provided with the specific circuit used to implement the oracle for the Grover search, there may be potential to optimize the classical algorithm to approach the theoretical complexity of the quantum algorithm. This concept falls under the umbrella of quantum-inspired classical algorithms [57]. For future work, we aim to explore the possibility of designing classical algorithms inspired by the Grover search-based approach in this paper to achieve a subcubic solution for DTC-based analysis.

8 RELATED WORK

We limit our discussion to the most relevant work to this paper, including DTC-based static program analysis and quantum speedups on classical problems.

Static program analysis. This paper aims to improve the worst-case time complexity of two typical static analysis algorithms, CFL-reachability and set constraint-based analysis, which are interchangeable and both based on dynamic transitive closure resulting in the algorithm having cubic complexity, known as a cubic bottleneck [34, 40, 42]. As far as we know, the best upper bound for CFL-reachability analysis is subcubic (i.e., $O(N^3/\log N)$) [15], but there is no truly subcubic solution. The algorithms with improved complexity only exist in the special case of DTC-based analysis. For example, a special case of CFL-reachability working on the Dyck language can be solved using bidirected trees and graphs in linear time [64]. For the BMM application, there is work that shows that solving Dyck-CFL-reachability on general graphs is BMM hard [14]. Another work shows the existence of subcubic certificate systems for CFL-reachability [20]. Developing faster algorithms in classical computing for general DTC-based analysis is highly challenging [51, 64]. This paper takes one step forward in investigating quantum speedups on the cubic bottleneck of DTC-based analysis. In recent years, the combination of classical program analysis and quantum computing has received much attention. However, most of them focus on extending classical techniques to solve problems in quantum programming theory (e.g., abstract interpretation [62], verification [66] and debugging of quantum programs [39]). We believe that the opposite direction, that is, using techniques and algorithms of quantum computing to tackle classical problems in programming languages, is equally important. We hope that this paper provides preliminary evidence that this goal is achievable, and how.

Quantum speedups. Recently, quantum computing has received much attention due to its substantial computational power. In 2022, the world's most extensive quantum computer had 433 qubits, and the 1000-qubit barrier will likely be broken in the near future [27]. Since hardware development is well-progressed, some quantum algorithms will likely be used in the coming years. Since hardware development is well-progressed, some quantum algorithms will likely be used in the coming years. Shor's algorithm [45, 50] solves integer factorization in polynomial time, whereas the best-known classical algorithm solves it in exponential time. Grover's algorithm [30] offers a quadratic speedup on unsorted search problems, whereas the classical algorithm needs linear time. These are two well-known algorithms in quantum computing. Quantum counting [11] is to estimate the number of targets before searching and may be an alternative solution to our approach. We did not use it in our approach because it is based on phase estimation [43] and requires much more complex oracles than we need in our quantum search. Specifically, a series of controlled versions of the currently used oracle is required, which will significantly increase the complexity. The recent implementation of data structures in quantum superposition [63] involves a quantum set based on a radix tree and QRAM, serving as an alternative to a quantum oracle. Unfortunately, most quantum programming languages lack support for the set data structure. Algorithms in quantum machine learning [9], chemistry [13], and quantum approximate optimization [25] have also been developed in the past years. Some applications of these algorithms are proven to have an effect on existing problems. RSA is possibly corrupted by Shor's algorithm [28]. Graph problems are improved by Grover's search, for example, the minimum spanning tree, connectivity, single-source shortest path [23], breadth-first search, and depth-first search [7].

9 CONCLUSION

This paper presents a quantum approach to speed up DTC-based static analysis algorithms. Its novelty lies in accelerating the key searching tasks by applying Grover's search when computing

dynamic transitive closures. To apply the quantum search subroutine, our quantum algorithm leverages QRAM to implement the oracle for loading classical data into quantum superposition. Due to the probabilistic nature of the quantum search, we further improve the existing quantum search subroutine with a probability amplification technique. Taking two static analysis algorithms as our applications, we prove that CFL-reachability and set constraint-based analysis have time complexity $O(N^2\sqrt{N}polylog(N))$, which is truly subcubic, outperforming the best upper bound. Our experiment results demonstrate the effectiveness and correctness of our approach. We hope our approach sheds light on new opportunities to address general challenging problems in static analysis using quantum computing.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their reviews and suggestions. This research is supported by Australian Research Grants DP210101348, FT220100391 and DP220102059, and by a generous Aspire Gift Grant from Google.

REFERENCES

[1] 2024. Artifact: "Dynamic Transitive Closure-Based Static Analysis through the Lens of Quantum Search". https://github.com/jiawei-95/tosem-QDTCSA-artifact

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1968. Time and Tape Complexity of Pushdown Automaton Languages. *Information and Control* 13 (1968), 186–206. https://doi.org/10.1016/S0019-9958(68)91087-5

[3] A. Aiken and E.L. Wimmers. 1992. Solving Systems of Set Constraints. In *1992 Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science*. 329–340. https://doi.org/10.1109/LICS.1992.185545

[4] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2562111

[5] Andris Ambainis. 2005. Quantum Search Algorithms. arXiv:quant-ph/0504012 [quant-ph]

[6] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgenijs Vihrovs. 2019. *Quantum Speedups for Exponential-Time Dynamic Programming Algorithms*. 1783–1793. https://doi.org/10.1137/1.9781611975482.107

[7] Andris Ambainis and Robert Špalek. 2006. Quantum Algorithms for Matching and Network Flows. In *Proceedings of the 23rd Annual Conference on Theoretical Aspects of Computer Science* (Marseille, France) *(STACS'06)*. Springer-Verlag, Berlin, Heidelberg, 172–183. https://doi.org/10.1007/11672142_13

[8] Lars Ole Andersen. 1994. Program Analysis and Specialization for the C Programming Language. (1994). https://www.cs.cornell.edu/courses/cs711/2005fa/papers/andersen-thesis94.pdf

[9] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum Machine Learning. *Nature* 549, 7671 (2017), 195–202. https://doi.org/10.1038/nature23474

[10] Rastisalv Bodík and Sadun Anik. 1998. Path-Sensitive Value-Flow Analysis. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '98)*. Association for Computing Machinery, New York, NY, USA, 237–251. https://doi.org/10.1145/268946.268966

[11] Michel Boyer, Gilles Brassard, Peter Hoyer, and Alain Tappa. 2005. Tight Bounds on Quantum Searching. *Fortschritte Der Physik-progress of Physics - FORTSCHR PHYS* 46 (01 2005), 187 – 199. https://doi.org/10.1002/(SICI)1521-3978(199806)46:

4/5%3C493::AID-PROP493%3E3.0.CO;2-P

[12] Harry Buhrman, Richard Cleve, Ronald de Wolf, and Christof Zalka. 1999. Bounds for Small-Error and Zero-Error Quantum Algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*. IEEE Computer Society, USA, 358. https://doi.org/10.1109/SFFCS.1999.814607

[13] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. 2019. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews* 119, 19 (2019), 10856–10915. https://doi.org/10.1021/acs.chemrev.8b00803

[14] Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2017. Optimal Dyck Reachability for Data-Dependence and Alias Analysis. *Proceedings of the ACM on Programming Languages* 2, POPL, Article 30 (dec 2017), 30 pages. https://doi.org/10.1145/3158118

[15] Swarat Chaudhuri. 2008. Subcubic Algorithms for Recursive State Machines. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, USA) *(POPL '08)*. Association for Computing Machinery, New York, NY, USA, 159–169. https://doi.org/10.1145/1328438.1328460

[16] Xiao Cheng, Xu Nie, Ningke Li, Haoyu Wang, Zheng Zheng, and Yulei Sui. 2022. How About Bug-Triggering Paths? - Understanding and Characterizing Learning-Based Vulnerability Detectors. *TDSC* (2022). https://doi.org/10.1109/TDSC.2022.3192419

[17] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. DeepWukong: Statically Detecting Software Vulnerabilities Using Deep Graph Neural Network. *TOSEM* (2021). https://doi.org/10.1145/3436877

[18] Xiao Cheng, Guanqin Zhang, Haoyu Wang, and Yulei Sui. 2022. Path-Sensitive Code Embedding via Contrastive Learning for Software Vulnerability Detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '22)*. ACM. https://doi.org/10.1145/3533767.3534371

[19] G. Chiribella, G. M. D'Ariano, and P. Perinotti. 2008. Quantum Circuit Architecture. *Physical Review Letters* 101 (Aug 2008), 060401. Issue 6. https://doi.org/10.1103/PhysRevLett.101.060401

[20] Dmitry Chistikov, Rupak Majumdar, and Philipp Schepper. 2022. Subcubic Certificates for CFL Reachability. *Proceedings of the ACM on Programming Languages* 6, POPL, Article 41 (jan 2022), 29 pages. https://doi.org/10.1145/3498702

[21] John Cortese and Timothy Braje. 2018. Loading Classical Data into a Quantum Computer. *ArXiv: Quantum Physics* (2018). https://doi.org/10.48550/arXiv.1803.01958

[22] Sebastian Dörn. 2008. Quantum Complexity of Graph and Algebraic Problems. (2008). https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.190/Mitarbeiter/doern/Dissertation.pdf

[23] Christoph Dürr, Mark Heiligman, Peter Hoyer, and Mehdi Mhalla. 2006. Quantum Query Complexity of Some Graph Problems. *SIAM J. Comput.* 35, 6 (2006), 1310–1328. https://doi.org/10.1137/050644719

[24] Sam Estep, Jenna Wise, Jonathan Aldrich, Éric Tanter, Johannes Bader, and Joshua Sunshine. 2021. Gradual Program Analysis for Null Pointers. In *Proceedings of the 35th European Conference on Object-Oriented Programming (ECOOP 2021) (Leibniz International Proceedings in Informatics (LIPIcs))*, Manu Sridharan and Anders Møller (Eds.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Aarhus, Denmark. https://doi.org/10.4230/LIPIcs.ECOOP.2021.3

[25] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A Quantum Approximate Optimization Algorithm. *ArXiv: Quantum Physics* (2014). https://doi.org/10.48550/arXiv.1411.4028

[26] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. 1987. The Program Dependence Graph and Its Use in Optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 9, 3 (jul 1987), 319–349. https://doi.org/10.1145/24039.24041

[27] Jay Gambetta. 2022. Quantum-Centric Supercomputing: The Next Wave of Computing. *IBM Research Blog* (2022). https://research.ibm.com/blog/next-wave-quantum-centric-supercomputing

[28] Craig Gidney and Martin Ekerå. 2021. How to Factor 2048 Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits. *Quantum* 5 (April 2021), 433. https://doi.org/10.22331/q-2021-04-15-433

[29] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. 2008. Quantum Random Access Memory. *Physical Review Letters* 100 (Apr 2008), 160501. Issue 16. https://doi.org/10.1103/PhysRevLett.100.160501

[30] 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

[31] Kathrin Hanauer, Monika Henzinger, and Christian Schulz. 2020. Faster Fully Dynamic Transitive Closure in Practice. *ArXiv* (2020). https://doi.org/10.48550/arXiv.2002.00813

[32] Charles Harris, K Millman, Stéfan Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten Kerkwijk, Matthew Brett, Allan Haldane, Jaime Río, Mark Wiebe, Pearu Peterson, and Travis Oliphant. 2020. Array programming with NumPy. *Nature* 585 (09 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[33] Nevin Heintze and Joxan Jaffar. 1994. Set Constraints and Set-Based Analysis. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming (PPCP '94)*. Springer-Verlag, Berlin, Heidelberg, 281–298. https://link.springer.com/chapter/10.1007/3-540-58601-6_107

[34] Nevin Heintze and David McAllester. 1997. On the Cubic Bottleneck in Subtyping and Flow Analysis. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS '97)*. IEEE Computer Society, USA, 342. https://doi.org/10.1109/LICS.1997.614960

[35] Asim Kadav, Matthew J. Renzelmann, and Michael M. Swift. 2009. Tolerating Hardware Device Failures in Software. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (Big Sky, Montana, USA) *(SOSP '09)*. Association for Computing Machinery, New York, NY, USA, 59–72. https://doi.org/10.1145/1629575.1629582

[36] Ioannis Krommidas and Christos Zaroliagis. 2008. An Experimental Study of Algorithms for Fully Dynamic Transitive Closure. *Journal of Experimental Algorithmics (JEA)* 12, Article 16 (jun 2008), 22 pages. https://doi.org/10.1145/1227161.1370597

[37] Yuxiang Lei, Yulei Sui, Shuo Ding, and Qirun Zhang. 2022. Taming Transitive Redundancy for Context-Free Language Reachability. In *OOPSLA*. https://doi.org/10.1145/3563343

[38] Yuxiang Lei, Yulei Sui, Shin Hwei Tan, and Qirun Zhang. 2023. Recursive State Machine Guided Graph Folding for Context-Free Language Reachability. *PLDI* (2023). https://doi.org/10.1145/3591233

[39] 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

[40] Anders Alnor Mathiasen and Andreas Pavlogiannis. 2021. The Fine-Grained and Parallel Complexity of Andersen's Pointer Analysis. *Proceedings of the ACM on Programming Languages* 5, POPL, Article 34 (jan 2021), 29 pages. https://doi.org/10.1145/3434315

[41] Olivia Di Matteo, Vlad Gheorghiu, and Michele Mosca. 2020. Fault-Tolerant Resource Estimation of Quantum Random-Access Memories. *IEEE Transactions on Quantum Engineering* 1 (2020), 1–13. https://doi.org/10.1109/TQE.2020.2965803

[42] David Melski and Thomas Reps. 1997. Interconvertbility of Set Constraints and Context-Free Language Reachability. In *Proceedings of the 1997 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation* (Amsterdam, The Netherlands) *(PEPM '97)*. Association for Computing Machinery, New York, NY, USA, 74–89. https://doi.org/10.1145/258993.259006

[43] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press. https://doi.org/10.1017/CBO9780511976667

[44] Alexandru Paler, Oumarou Oumarou, and Robert Basmadjian. 2020. Parallelizing the Queries in a Bucket-Brigade Quantum Random Access Memory. *Physical Review A* 102 (Sep 2020), 032608. Issue 3. https://doi.org/10.1103/PhysRevA.102.032608

[45] Yuxiang Peng, Kesha Hietala, Runzhou Tao, Liyi Li, Robert Rand, Michael Hicks, and Xiaodi Wu. 2022. A Formally Certified End-to-End Implementation of Shor's Factorization Algorithm. https://doi.org/arXiv.2204.07112

[46] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (01 2018). https://doi.org/10.22331/q-2018-08-06-79

[47] Daniel Reitzner and Mark Hillery. 2019. Grover Search Under Localized Dephasing. *Phys. Rev. A* 99 (Jan 2019), 012339. Issue 1. https://doi.org/10.1103/PhysRevA.99.012339

[48] Thomas Reps. 1997. Program Analysis via Graph Reachability. In *Proceedings of the 1997 International Symposium on Logic Programming* (Port Washington, New York, USA) *(ILPS '97)*. MIT Press, Cambridge, MA, USA, 5–19. https://doi.org/10.1016/S0950-5849(98)00093-7

[49] Qingkai Shi, Xiao Xiao, Rongxin Wu, Jinguo Zhou, Gang Fan, and Charles Zhang. 2018. Pinpoint: Fast and Precise Sparse Value Flow Analysis for Million Lines of Code. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. Association for Computing Machinery, New York, NY, USA, 693–706. https://doi.org/10.1145/3192366.3192418

[50] 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

[51] Bjarne Steensgaard. 1996. Points-to Analysis in Almost Linear Time. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (St. Petersburg Beach, Florida, USA) *(POPL '96)*. Association for Computing Machinery, New York, NY, USA, 32–41. https://doi.org/10.1145/237721.237727

[52] E. M. Stoudenmire and Xavier Waintal. 2023. Grover's Algorithm Offers No Quantum Advantage. https://doi.org/10.48550/arXiv.2303.11317

[53] Yulei Sui, Xiao Cheng, Guanqin Zhang, and Haoyu Wang. 2020. Flow2Vec: Value-Flow-Based Precise Code Embedding. *OOPSLA* (2020). https://doi.org/10.1145/3428301

[54] Yulei Sui and Jingling Xue. 2016. SVF: Interprocedural Static Value-Flow Analysis in LLVM. In *Proceedings of the 25th International Conference on Compiler Construction* (Barcelona, Spain) *(CC 2016)*. Association for Computing Machinery,

New York, NY, USA, 265–266. https://doi.org/10.1145/2892208.2892235

[55] Yulei Sui, Ding Ye, and Jingling Xue. 2012. Static Memory Leak Detection Using Full-Sparse Value-Flow Analysis. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis* (Minneapolis, MN, USA) *(ISSTA 2012)*. Association for Computing Machinery, New York, NY, USA, 254–264. https://doi.org/10.1145/2338965.2336784

[56] Yulei Sui, Ding Ye, and Jingling Xue. 2014. Detecting Memory Leaks Statically with Full-Sparse Value-Flow Analysis. *TSE* (2014). https://doi.org/10.1109/TSE.2014.2302311

[57] Ewin Tang. 2019. A Quantum-Inspired Classical Algorithm for Recommendation Systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Phoenix, AZ, USA) *(STOC 2019)*. Association for Computing Machinery, New York, NY, USA, 217–228. https://doi.org/10.1145/3313276.3316310

[58] Thomas Wigren. 2015. The Cauchy-Schwarz inequality : Proofs and applications in various spaces. https://www.diva-portal.org/smash/get/diva2:861242/FULLTEXT02.pdf

[59] Virginia Vassilevska Williams. 2012. Multiplying Matrices Faster than Coppersmith-Winograd. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '12)*. Association for Computing Machinery, New York, NY, USA, 887–898. https://doi.org/10.1145/2213977.2214056

[60] Cheng Xiao, Wang Jiawei, and Sui Yulei. 2024. Precise Sparse Abstract Execution via Cross-Domain Interaction. In *46th International Conference on Software Engineering (ICSE '2024)*. ACM/IEEE. https://doi.org/10.1145/3597503.3639220

[61] Mihalis Yannakakis. 1990. Graph-Theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Nashville, Tennessee, USA) *(PODS '90)*. Association for Computing Machinery, New York, NY, USA, 230–242. https://doi.org/10.1145/298514.298576

[62] Nengkun Yu and Jens Palsberg. 2021. Quantum Abstract Interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (Virtual, Canada) *(PLDI 2021)*. Association for Computing Machinery, New York, NY, USA, 542–558. https://doi.org/10.1145/3453483.3454061

[63] Charles Yuan and Michael Carbin. 2022. Tower: Data Structures in Quantum Superposition. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 134 (oct 2022), 30 pages. https://doi.org/10.1145/3563297

[64] Qirun Zhang, Michael R. Lyu, Hao Yuan, and Zhendong Su. 2013. Fast Algorithms for Dyck-CFL-Reachability with Applications to Alias Analysis. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Seattle, Washington, USA) *(PLDI '13)*. Association for Computing Machinery, New York, NY, USA, 435–446. https://doi.org/10.1145/2491956.2462159

[65] Xin Zheng and Radu Rugina. 2008. Demand-Driven Alias Analysis for C. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 197–208. https://doi.org/10.1145/1328897.1328464

[66] Li Zhou, Gilles Barthe, Pierre-Yves Strub, Junyi Liu, and Mingsheng Ying. 2023. CoqQ: Foundational Verification of Quantum Programs. 7, POPL, Article 29 (jan 2023), 33 pages. https://doi.org/10.1145/3571222

[67] Yiqing Zhou, E Miles Stoudenmire, and Xavier Waintal. 2020. What Limits the Simulation of Quantum Computers? *Physical Review X* 10, 4 (2020), 041038. https://doi.org/10.1103/PhysRevX.10.041038