10-13-2024

A Concept of Controlling Grover Diffusion Operator: a New Approach to Solve Arbitrary Boolean-Based Problems

Ali Al-Bayaty
Portland State University, albayaty@pdx.edu

Marek A. Perkowski
Portland State University, h8mp@pdx.edu

Citation Details

Al-Bayaty, A., & Perkowski, M. (2024). A concept of controlling Grover diffusion operator: a new approach to solve arbitrary Boolean-based problems. Scientific Reports, 14(1).

scientific reports

Check for updates

OPEN

A concept of controlling Grover diffusion operator: a new approach to solve arbitrary Boolean-based problems

Ali Al-Bayaty & Marek Perkowski

A controlled-diffusion operator for Boolean oracles is designed as a new approach for Grover's algorithm to search for solutions for arbitrary logical structures of such oracles, since the Grover diffusion operator is not able to find correct solutions for some logical structures of Boolean oracles. We also show that the Phase oracles do not work sometimes correctly using the Grover diffusion operator. Our proposed controlled-diffusion operator relies on the states of output qubit, as the reflection of Boolean decisions from a Boolean oracle without relying on the phase kickback. We prove that on many examples of Boolean and Phase oracles the Grover diffusion operator is not working correctly. The oracles in these examples are constructed using different structures of POS, SOP, ESOP, CSP-SAT, and XOR-SAT. Our mathematical models and experiments prove that the proposed controlled-diffusion operator successfully searches for all solutions for all Boolean oracles regardless of their different logical structures.

Keywords Grover's algorithm, Grover diffusion operator, Controlled-diffusion operator, Boolean oracle, Phase oracle, Logical structures

Grover's algorithm[1–5] is the most well-known quantum search algorithm to find solutions for both Boolean and Phase oracles with quadratic speedup, and to construct other quantum algorithms, such as the quantum counting algorithm[4,6–8]. The papers of L. K. Grover[1,2] introduced Grover's algorithm for the Phase oracles, and his paper[3] presented Grover's algorithm for the Boolean oracles. Although the quantum circuit of a Phase oracle usually utilizes fewer qubits and requires less quantum gates than the quantum circuit of a Boolean oracle[9], the Boolean oracles are very useful because they are reversible binary circuits that can be easily converted from classical Boolean circuits. In the classical domain, an NP-complete problem[4,5,10] can be expressed as a sequential exercise of a classical Boolean oracle. The designer can construct a classical Boolean oracle using different Boolean structures, such as Product-Of-Sums (POS)[11], Sum-Of-Products (SOP)[12], Exclusive-or Sum-Of-Products (ESOP)[13–16], XOR-Satisfiability (XOR-SAT)[17–19], just to name a few. These Boolean structures utilize classical Boolean gates, e.g., NOT, AND, OR, etc. Classical Boolean oracles are then transformed into their equivalent quantum Boolean oracle using reversible quantum gates. There are many known methods to synthesize and optimize quantum reversible circuits. For instance, the reversible quantum gates for a quantum Boolean oracle can be designed based on their classical truth tables, De Morgan's Laws[20], Algebraic Normal Form (ANF) (or Reed-Muller expansion)[21–23], and ESOP synthesis[13–16], depending on the functionality of a problem. The designer familiar with classical digital systems uses hierarchical blocks, such as arithmetic circuits, comparators, counters, etc., to design the Boolean reversible circuit of a quantum Boolean oracle.

When the reversible quantum gates are designed for flipping the states of input qubits, i.e., from $|0\rangle$ to $|1\rangle$ or vice versa, then the final quantum oracle is termed the "Boolean oracle". However, when the reversible quantum gates are designed for inverting the phases of input qubits (q), i.e., from $+|q\rangle$ to $-|q\rangle$ or vice versa, then the final quantum oracle is termed the "Phase oracle". Hence, a Boolean oracle is built from "quantum Boolean-based gates", while a Phase oracle is built from "quantum phase-based gates". The Boolean or Phase oracle is then evaluated using Grover's algorithm, in the evaluation complexity of $O\left(\sqrt{N}\right)$ for one solution (one marked element) or in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ for k solutions (k marked elements). The evaluation complexity is the so-called

Department of Electrical and Computer Engineering, Portland State University, Portland, USA. email: albayaty@pdx.edu

"Grover iterations" or "Grover loops". In binary quantum computing, $N=2^n$, where n is the total number of input qubits for an oracle that Grover's algorithm utilizes them to solve a problem.

In general, Grover's algorithm consists of three components, which we termed them the "Blocks" in this article:

1. "Block$_1$" initializes n input qubits to a uniform distribution using Hadamard (H) gates. The H gates impose all n input qubits into uniform superposition states, to generate a complete quantum search space of $\{|0\rangle, |1\rangle\}^{\otimes n}$ for Grover's algorithm to search for marked elements (solutions). The H gates are considered as an implicit generator for Grover's algorithm.
2. "Block$_2$" consists of a Boolean or Phase oracle that adds negative phases to marked elements, i.e., inverts the phases of vectors of input qubits as the "first rotation of solutions" over the complete quantum search space. Such phase inversion occurs due to the phase kickback for a Boolean oracle, or due to the effect of quantum phase-based gates on the input qubits for a Phase oracle.
3. "Block$_3$" consists of the Grover diffusion operator that performs the "second rotation of solutions", through the conditional phase shift and phase inversion, by rotating and amplifying the amplitudes of the marked elements from Block$_2$. Note that both Block$_2$ and Block$_3$ are treated as one Grover iteration.

For a Boolean oracle, the output qubit ensures the phase kickback on the marked elements, when the output qubit is initially set to the state of $|-\rangle$. In our article, the output qubit is termed the "functional qubit (fqubit)". However, neither the fqubit nor the phase kickback is utilized for a Phase oracle, since the inversion of phases is implicitly performed using quantum phase-based gates, e.g., Pauli-Z gate (Z), controlled-Z gate (CZ), and similar gates. For instance, to solve a problem using Grover's algorithm[1–5] with Block$_1$ in one Grover iteration, we need the following:

- The Boolean oracle (U_B) of input qubits (q) represents a problem by utilizing the fqubit as an output qubit; such that $f\,(q, fqubit) : U_B\,|q\,,\,fqubit\rangle \mapsto |q\,,\,fqubit \oplus f\,(q)\rangle$, i.e., the f($q$) is encoded in the fqubit as computational basis states of $|0\rangle$ (a non-solution) and $|1\rangle$ (a solution).
- The phase kickback reflects the solution of U_B in the encoded states of fqubit to the input qubits as phase inversion; such that $f\,(q, -) : U_B\,|q\,,\,-\rangle \mapsto (-1)^{f(q)}\,|q\,,\,-\rangle$, i.e., the phase kickback occurs when the fqubit initially sets to the state of $|-\rangle$, and the solution is now represented by the inverted phase (–) on the vectors of input qubits (as one marked element).
- The Phase oracle (U_P) of input qubits (q) represents a problem without utilizing the fqubit and the phase kickback; such that $f\,(q) : U_P\,|q\,\rangle \mapsto (-1)^{f(q)}\,|q\,\rangle$, i.e., the solution is directly represented by the inverted phase (–) on the vectors of input qubits (as one marked element).
- The Grover diffusion operator rotates and amplifies the amplitude of the marked element, in which it is then measured as the highest probability as the solution for a problem, in the classical domain.

For a complete Gover's algorithm, Block$_1$ and all $O\left(\sqrt{N}\right)$ Grover iterations (repetitions of Block$_2$ and Block$_3$) are applied at once to search for one marked element, and then all n input qubits are classically measured to observe the highest probability as one solution[1–3]. However, in our modified method of Grover's algorithm, we apply Block$_1$ and one Grover iteration, if a solution is not observed after measurement, then Block$_1$ and two Grover iterations are re-applied, and so on until all k solutions are observed in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ for $k \geq 1$, as illustrated in Fig. 1a. Moreover, the components of the U_s operator are operationally demonstrated in Fig. 1b.

In the terminology of Grover's algorithm, (i) a Boolean or Phase oracle is denoted as U_ω, where U states for "unitary" and ω states for "winner" (as a marked element), and (ii) Grover diffusion operator is denoted as U_s, where s states for "search". Such that, a U_ω evaluates a problem for one marked element and a number of unmarked elements. All marked and unmarked elements have equally normalized amplitudes $\left(\frac{1}{\sqrt{N}}\right)$. Thereafter, the U_s operator searches only for the marked element by rotating and amplifying its amplitude $\left(> \frac{1}{\sqrt{N}}\right)$ as well as decreasing the amplitudes of unmarked elements $\left(< \frac{1}{\sqrt{N}}\right)$, in one Grover iteration[1–5].

Before the U_s operator rotates and amplifies the amplitude of a marked element, a Boolean or Phase oracle (U_ω) needs to be uncomputed to ensure that all its ancillae are reset to their initial states of $|0\rangle$. Resetting the ancillae ensures to re-utilize them for further Grover iterations, and such a resetting is achieved by reversing (mirroring) the U_ω, except for its "collector gate". Note that (i) if a U_ω (Boolean or Phase oracle) does not utilize any ancillae in its design, then the mirroring is not needed, and (ii) the collector gate of a U_ω (Phase oracle only) is implicitly fused with the other quantum phase-based gates depending on the design implementation[9,24]. For that, the total number of utilized qubits decreases for Phase oracles and increases for Boolean oracles. Depending on the design implementation, Phase oracles are mainly utilizing n input qubits with almost no ancillae, while Boolean oracles are mostly utilizing n input qubits, m ancilla qubits (ancillae), and one fqubit, where $n \geq 2$ and $m \geq 0$.

***Definition 1*:** The collector gate is the junction gate(s) associated with the clause(s) of literal(s) for a Boolean structure that defines a Boolean oracle. Such that, (i) the collector gate for a Boolean structure in conjunctive normal form (CNF)[25], also called POS, is the conjunction AND gate(s) for the disjunction OR clause(s) of literal(s), (ii) the collector gate for a Boolean structure in disjunctive normal form (DNF)[26], also called SOP, is the disjunction OR gate(s) for the conjunction AND clause(s) of literal(s), (iii) the collector gate for a Boolean

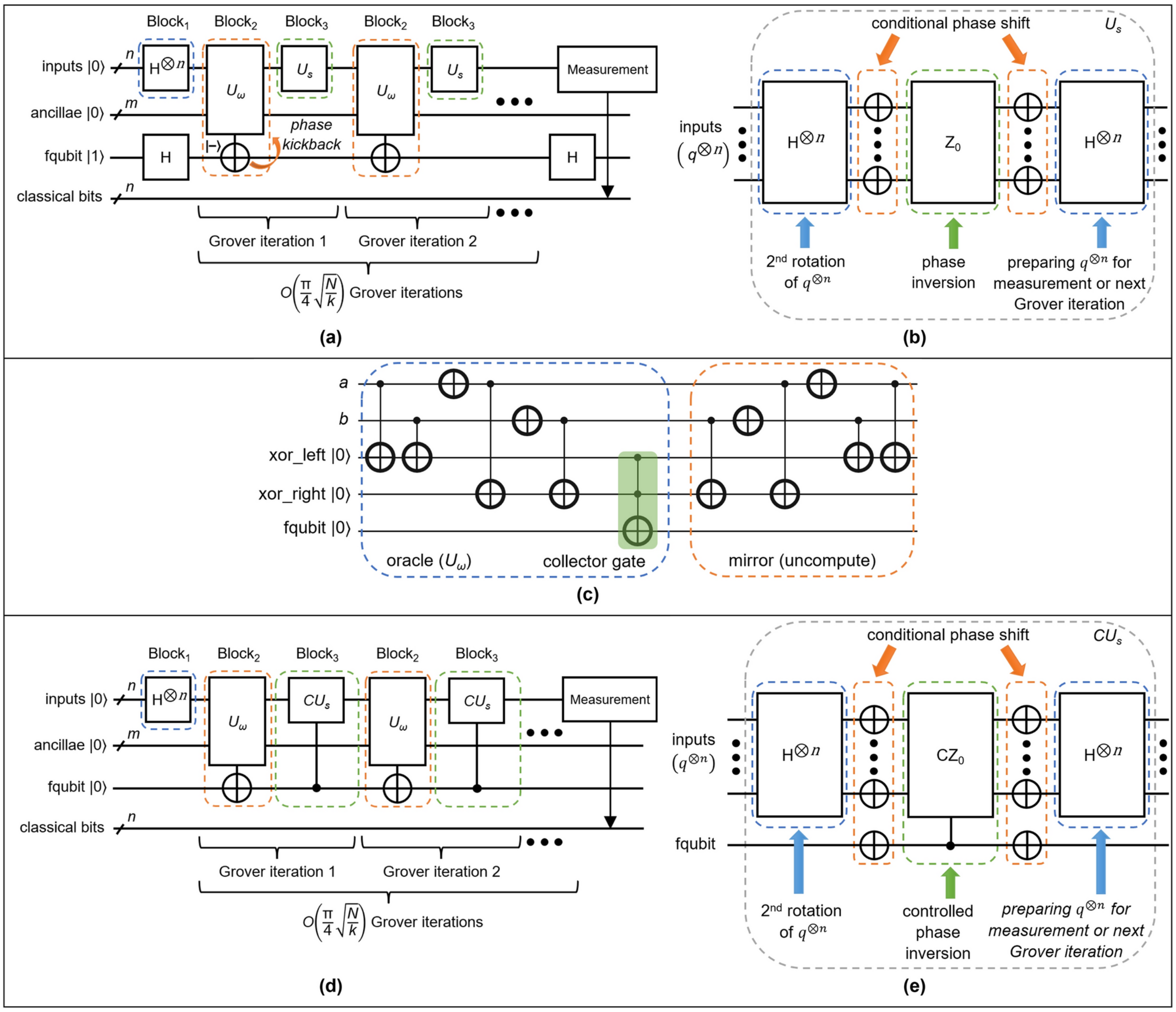

Fig. 1. Schematics of Grover's algorithms, Boolean oracles, a collector gate, and quantum operators: (**a**) Grover's algorithm for n input qubits, m ancilla qubits, one functional qubit (fqubit), n classical bits (for final measurement), Block$_1$ consists of H gates for n input qubits, Block$_2$ is a Boolean oracle (U_ω), Block$_3$ is Grover diffusion operator (U_s), and the number of Grover iterations, (**b**) the components and functionalities of Grover diffusion operator (U_s) for the second rotation operation (H), conditional phase shift (X), and phase inversion (Z_0), (**c**) the quantum Boolean-based problem in the Boolean structure of $(a \oplus b) \wedge (\neg a \oplus \neg b)$, as a Boolean oracle ($U_\omega$), the collector gate is in the form of Boolean AND gate (Toffoli in green), and the mirror (uncomputing part) of U_ω, (**d**) Grover's algorithm for n input qubits, m ancilla qubits, one fqubit, n classical bits (for final measurement), Block$_1$ consists of H gates for n input qubits, Block$_2$ is a Boolean oracle (U_ω), Block$_3$ is our proposed controlled-diffusion operator (CU_s), and the number of Grover iterations, and (**e**) the components and functionalities of our controlled-diffusion operator (CU_s) for the second rotation operation (H), conditional phase shift (X), and controlled phase inversion (CZ_0) by the states of fqubit. Where $n \geq 2$, $m \geq 0$, $N = 2^n$, and $k \geq 1$ solution (marked element). Note: Grover's algorithm for a Phase oracle can be constructed similarly as in (**a**), without the fqubit and the number of ancillae varies depending on the circuit design of a Phase oracle.

structure in ESOP is the XOR gate(s) for the conjunction AND clause(s) of literal(s), just to name a few. The literals are the inputs of the Boolean structure for a Boolean oracle.

For instance, the Boolean oracle (U_ω) for a problem in the Boolean structure of $(a \oplus b) \wedge (\neg a \oplus \neg b)$ consists of (i) the conjunctive AND gate, i.e., the collector gate as Toffoli gate, and (ii) two XOR clauses for the literals (a and b), as illustrated in Fig. 1c, where the U_ω is surrounded by the blue rectangle, the collector gate is located in the green area, and the mirror (uncompute) is surrounded by the orange rectangle.

In this article, our work shows that Grover diffusion operator (U_s) successfully searches for solutions when the collector gate for a Boolean oracle (U_ω) is expressed in the form of Boolean AND gate, for the Boolean structures of AND-XOR, ESOP, and a few of POS! However, the U_s operator fails to search for solutions when the collector gate for a Boolean oracle (U_ω) is expressed in different Boolean gates, such as NAND, OR, or XNOR, for different Boolean structures, e.g., OR-XOR and SOP. That means the phase kickback does not effectively invert the phase of a marked element, as a solution, for a Boolean oracle built from quantum Boolean-based gates. Similarly, the U_s operator either succeeds or fails for a Phase oracle equivalent to a Boolean oracle for the same problem!

For such a reason, our work redesigns Grover diffusion operator (U_s) to the controlled-diffusion operator that searches for solutions based on the computational basis states of fqubit, i.e., $|0\rangle$ and $|1\rangle$, instead of relying on the phase kickback, for Boolean oracles only. Our controlled-diffusion operator is substantially the U_s operator that is controlled by the states of fqubit. The states of fqubit act as the controlling signals that inform the U_s operator to distinguish between the unmarked element (a non-solution) and the marked element (a solution). For the controlled-diffusion operator, the fqubit is initially set to the state of $|0\rangle$ instead of $|-\rangle$, so the collector gate of a Boolean oracle (U_ω) just flips a state of fqubit from $|0\rangle$ to $|1\rangle$ to indicate a marked element, or the state of fqubit remains at $|0\rangle$ to indicate an unmarked element. Both indications are then utilized to signal the controlled-diffusion operator.

Consequently, our proposed controlled-diffusion operator is denoted as the "Controlled-U_s (CU_s)" operator. As mathematically and experimentally proved in this article (and in the "Supplementary Information"), the CU_s operator successfully searches for all k solutions for arbitrary Boolean oracles regardless of their (i) different Boolean structures and (ii) different Boolean forms of collector gates, in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ Grover iterations for $k \geq 1$.

For Grover's algorithm, the CU_s operator is designed to search for all k solutions for Boolean oracles based on the states of fqubit. However, the CU_s operator is not designed for Phase oracles since the fqubit is not utilized in their quantum circuits design. Note that, for a Boolean oracle (U_ω), (i) the U_s operator searches for solutions by rotating and amplifying the amplitudes of marked elements throughout the reflection of phase kickback from the fqubit to U_ω, in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ Grover iterations, while (ii) our CU_s operator searches for solutions by rotating and amplifying the amplitudes of marked elements throughout the reflection of Boolean decisions from the collector gate of U_ω to the states of fqubit, in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ Grover iterations.

The CU_s operator follows the same structural design of Grover's algorithm using the aforementioned three blocks, except for the fqubit that initially sets to the $|0\rangle$ state, since the phase kickback is not utilized in our design methodology. Figure 1d illustrates the workflow of Grover's algorithm of CU_s operator, and the components and functionalities of the CU_s operator are demonstrated in Fig. 1e. Note that, in Fig. 1e, Pauli-X gates of the conditional phase shift are added to fqubit to match the working philosophy of U_s operator as well, i.e., $CZ_0 \equiv Z_0$, (as it is discussed further in the "Supplementary Information").

Our work is concentrated on implementing Grover's algorithm of CU_s operator to search for all solutions for Boolean oracles, due to the reasons of (i) Boolean oracles are easier and more straightforward in design using quantum Boolean-based gates (X, CX, CCX) rather than quantum phase-based gates (Z, CZ, CCZ), (ii) the quantum Boolean-based gates can be directly realized using the truth tables or De Morgan's Laws for classical Boolean gates, (iii) the quantum Boolean-based gates and their qubits can be easily analyzed using classical Boolean Logic, i.e., a Boolean Logic of '0' represents a quantum state of $|0\rangle$ and a Boolean Logic of '1' represents a quantum state of $|1\rangle$, and (iv) Boolean oracles consists of collector gates that easily flip the states of fqubit to indicate the marked elements (solutions), while collector gates are not utilized in Phase oracles.

In this article, arbitrary problems are designed as Boolean and Phase oracles, and then these oracles are evaluated using Grover's algorithm of U_s operator (for both Boolean and Phase oracles) and using Grover's algorithm of CU_s operator (for Boolean oracles only). Subsequently, the two Grover's algorithms for U_s and CU_s operators, respectively, are compared in the manner of (i) their ability to search for all solutions and (ii) how many Grover iterations are required. These Boolean and Phase oracles are chosen to represent different applications in the Boolean structures of primitive Boolean gates (NOT, AND, NAND, OR, NOR, XOR, and XNOR), POS, SOP, ESOP, 1-bit ORed half-adder[27], 2-bit inequality comparator[27], 3-node graph coloring[28,29], and 2×2 Sudoku game[30,31]. Concluding that Grover's algorithm of U_s operator only solves Boolean oracles (and their equivalent Phase oracles), when their collector gates are in the form of Boolean AND gates for the Boolean structures of ESOP, AND-XOR, and a few of POS. While Grover's algorithm of CU_s operator successfully solves all Boolean oracles regardless of their Boolean structures and the Boolean forms of their collector gates. For solved applications, both Grover's algorithms of U_s and CU_s operators find similar solutions for the same Grover iterations.

Different research papers, for instance in[9,32–35], evaluated Boolean and Phase oracles using Grover's algorithm of U_s operator, in the manner of (i) reducing Grover iterations for $\leq O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$, and (ii) converting Boolean oracles to Phase oracles for a fewer number of quantum gates and qubits. To the best of our knowledge, all these propositions and implementations were performed using Grover's algorithm of U_s operator, which totally depends on the phase kickback for a Boolean oracle and the phase inversion for a Phase oracle. In[9,32–35], we noticed that all collector gates for Boolean oracles are constructed in the form of Boolean AND gate (as the Toffoli gate), and in the form of multiple-controlled Z gate (as the Toffoli-based) for Phase oracles. However, none of these papers realized the collector gates in other forms of Boolean gates, such as NAND and OR. Consequently, the following questions may arise when solving problems using Grover's algorithm of U_s operator:

1. How does Grover's algorithm solve an oracle problem formulated in the Boolean structure of SOP? Grover's algorithm of U_s operator is not able to solve the corresponding oracle in the structure of SOP!
2. How does Grover's algorithm solve any oracle problem formulated in any Boolean structure with NAND or NOR as a collector gate? Grover's algorithm of U_s operator is not able to solve any oracle in such structures!

In this article, our research focuses on solving arbitrary problems that are transformed into Boolean oracles of various Boolean forms of collector gates and in different Boolean structures. For that, we design and propose the controlled-diffusion operator (CU_s) for Grover's algorithm to successfully search for all k solutions for Boolean oracles only, in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ Grover iterations for $k \geq 1$ solution, as compared to such non-solvable problems when using Grover's algorithm of Grover diffusion operator (U_s) for Boolean and Phase oracles.

Results

In the classical domain, arbitrary problems are designed in the Boolean structures of classical Boolean gates, POS, SOP, ESOP, digital logic circuits, constraint satisfaction problems–satisfiability (CSP-SAT)[29,36], and XOR-SAT. Thereafter, these designed problems are transformed into their equivalent Boolean and Phase oracles (U_ω) using quantum Boolean-based gates and quantum Phase-based gates, respectively. To solve these designed problems and search for all their solutions in the quantum domain, their oracles of different collector gates, which are chosen based on the purpose of classical applications, are evaluated using Grover's algorithm of U_s operator for Boolean and Phase oracles and Grover's algorithm of CU_s operator for Boolean oracles only, in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ Grover iterations, where $N = 2$ (total no. of input qubits) and $k \geq 1$ solution.

In our work, the CU_s operator successfully searches for all solutions for Boolean oracles regardless of their various Boolean forms of collector gates. While the U_s operator only searches for solutions for Boolean and Phase oracles when their collector gates are in the form of Boolean AND gate, and it fails on other Boolean forms of collector gates. For ease of illustrating the purpose of our proposed CU_s operator as compared to the U_s operator, the Boolean and Phase oracles (Block$_2$) are only sketched without the other components (Block$_1$ and Block$_3$) of Grover's algorithm as well as the measurements. Note that the complete Grover's algorithms for both operators (U_s and CU_s) follow the previously illustrated constructions in Fig. 1a and Fig. 1d, respectively.

The IBM Quantum Platform was utilized to design Boolean and Phase oracles, implement both operators (U_s and CU_s), and perform Grover's algorithm to solve the problems. Grover's algorithm was simulated using the quantum simulator (ibmq_qasm_simulator) with 1024 resampling simulation times (as shots)[37]. However, the time of simulations (as a speed comparison factor) is not considered in this article, since such a factor varies and depends on (i) the current usage of this on-the-cloud simulator and (ii) the internet speed. The Boolean oracles are constructed using the truth tables and De Morgan's Laws of their transformed classical Boolean gates, while the Phase oracles are directly generated using the Qiskit instruction (PhaseOracle)[24].

Oracles of primitive boolean gates

The following Boolean and Phase oracles are designed as single collector gates using a primitive form of Boolean gates (NOT, AND, NAND, OR, NOR, XOR, and XNOR). The Boolean oracles of all primitive Boolean gates have two input qubits and one output qubit (fqubit), except for the Boolean NOT oracle that has one input qubit and one fqubit. Thus, these Boolean oracles imitate the same architecture of classical Boolean gates in the quantum domain.

(i) The Boolean oracle for the classical Boolean NOT gate is designed, as shown in Fig. 2a. The CU_s operator successfully searches for the solution, as shown in Fig. 2b. While the U_s operator cannot be constructed for such an architecture of classical Boolean NOT gate for Boolean and Phase oracles! Hence, the CU_s operator shows its advantage over the U_s operator in searching for solutions for further arbitrary Boolean oracles, when their collector gates are associated with the form of Boolean NOT gate, such as NAND, NOR, and XNOR. For this primitive Boolean gate, its Boolean oracle is generated using the Boolean structure of $(\neg a)$ based on NOT's truth table.
(ii) The Phase oracle for the classical Boolean AND gate is simply the controlled-Z gate (CZ), as illustrated in Fig. 2c, and its Boolean oracle is simply the Toffoli gate, as shown in Fig. 2d. The U_s operator successfully searches for the solution for the Phase and Boolean oracles, as demonstrated in Fig. 2e and Fig. 2f, respectively. Also, the CU_s operator successfully searches for the solution for the Boolean oracle, as presented in Fig. 2g. Boolean and Phase oracles are generated using the Boolean structure of $(a \wedge b)$ based on AND's truth table.
(iii) The Phase and Boolean oracles for the classical Boolean NAND gate are designed, as shown in Fig. 2h and Fig. 2i, respectively. The CU_s operator successfully searches for all solutions for the Boolean oracle, as illustrated in Fig. 2l, while the U_s operator fails to search for solutions for Phase and Boolean oracles, as shown in Fig. 2j and Fig. 2k, respectively. We noticed that the U_s operator treats these Phase and Boolean NAND oracles as Phase and Boolean AND oracles! Therefore, the CU_s operator shows its advantage over the U_s operator in searching for all solutions for further arbitrary Boolean oracles, when their collector gates are in the form of Boolean NAND gate. For this primitive Boolean gate, Boolean and Phase oracles are generated using the Boolean structure of $\neg(a \wedge b)$ based on NAND's truth table.
(iv) The Boolean and Phase oracles for the classical Boolean OR gate are designed using either the OR's truth table or De Morgan's Law. Note that both designs have the same ORing functionality. Phase and Boolean oracles are generated using the Boolean structure of $(a \vee b)$ based on OR's truth table, as shown in Fig. 3a and Fig. 3b, respectively. Moreover, the Boolean oracle can be designed using the Boolean structure of $\neg(\neg a \wedge \neg b)$ based on De Morgan's Laws, as presented in Fig. 3c. The CU_s operator successfully searches for all solutions

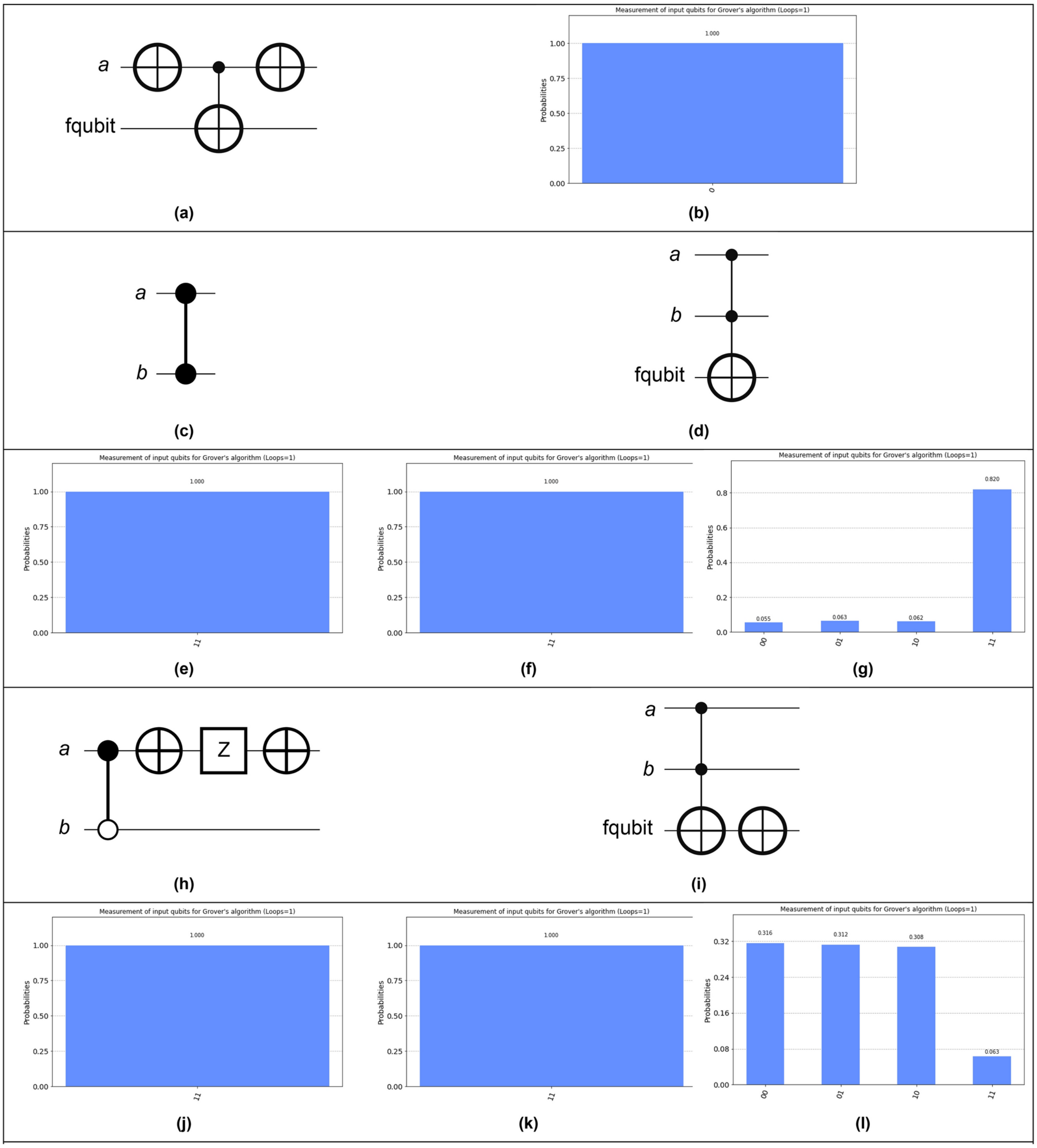
a
fqubit
(a)
Measurement of input qubits for Grover's algorithm (Loops=1)
Probabilities
1.000
0
(b)
a
b
(c)
a
b
fqubit
(d)
Measurement of input qubits for Grover's algorithm (Loops=1)
1.000
11
(e)
Measurement of input qubits for Grover's algorithm (Loops=1)
1.000
11
(f)
Measurement of input qubits for Grover's algorithm (Loops=1)
0.055
0.063
0.062
0.820
00
01
10
11
(g)
a
b
Z
(h)
a
b
fqubit
(i)
Measurement of input qubits for Grover's algorithm (Loops=1)
1.000
11
(j)
Measurement of input qubits for Grover's algorithm (Loops=1)
1.000
11
(k)
Measurement of input qubits for Grover's algorithm (Loops=1)
0.316
0.312
0.308
0.063
00
01
10
11
(l)

Fig. 2. Boolean and Phase oracles for classical primitive Boolean gates (NOT, AND, NAND) with the final measured solutions using the U_s and CU_s operators: (**a**) Boolean oracle of classical Boolean NOT gate, this Boolean gate cannot be implemented as Phase oracle, (**b**) the solution for Boolean NOT oracle using the CU_s operator, (**c**) Phase oracle of classical Boolean AND gate, (**d**) Boolean oracle of classical Boolean AND gate, (**e**) the solution for Phase AND oracle using the U_s operator, (**f**) the solution for Boolean AND oracle using the U_s operator, (**g**) the solution for Boolean AND oracle using the CU_s operator, (**h**) Phase oracle of classical Boolean NAND gate, (**i**) Boolean oracle of classical Boolean NAND gate, (**j**) the non-solution for Phase NAND oracle using the U_s operator, (**k**) the non-solution for Boolean NAND oracle using the U_s operator, and (**l**) the solutions for Boolean NAND oracle using the CU_s operator. Note: the CU_s operator successfully searches for all solutions for all Boolean oracles of classical primitive Boolean gates (NOT, AND, and NAND).

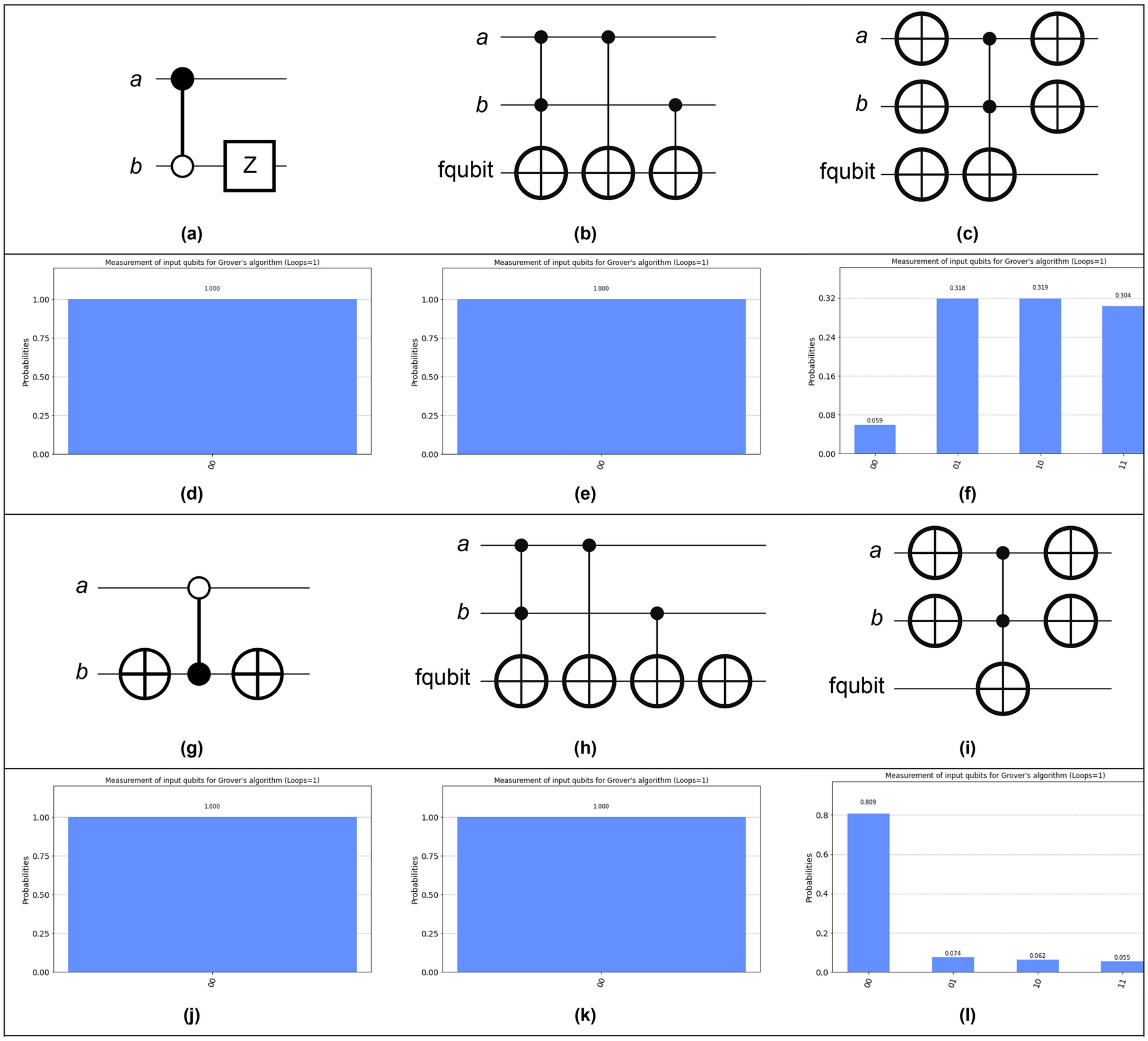

Fig. 3. Boolean and Phase oracles for classical primitive Boolean gates (OR and NOR) with the final measured solutions using the U_s and CU_s operators: (**a**) Phase oracle of classical Boolean OR gate, (**b**) Boolean oracle of classical Boolean OR gate from the OR's truth table, (**c**) Boolean oracle of classical Boolean OR gate from De Morgan's Law of $\neg(\neg a \wedge \neg b)$, (**d**) the non-solution for Phase OR oracle using the U_s operator, (**e**) the non-solution for Boolean OR oracle using the U_s operator, (**f**) the solutions for Boolean OR oracle using the CU_s operator, (**g**) Phase oracle of classical Boolean NOR gate, (**h**) Boolean oracle of classical Boolean NOR gate from the NOR's truth table, (**i**) Boolean oracle of classical Boolean NOR gate from De Morgan's Law of $(\neg a \wedge \neg b)$, (**j**) the solution for Phase NOR oracle using the U_s operator, (**k**) the solution for Boolean NOR oracle using the U_s operator, and (**l**) the solution for Boolean NOR oracle using the CU_s operator. Note: the CU_s operator successfully searches for all solutions for all Boolean oracles of classical primitive Boolean gates (OR and NOR).

for the Boolean oracle, as shown in Fig. 3f, while the U_s operator fails to search for solutions for Phase and Boolean oracles, as demonstrated in Fig. 3d and Fig. 3e, respectively. We noticed that the U_s operator treats these Boolean and Phase OR oracles as Boolean and Phase NOR oracles! Concluding that the CU_s operator shows its advantage over the U_s operator in searching for solutions for collector gates in the form of Boolean OR gate.

(v) The Boolean oracle for the classical Boolean NOR gate is designed using either the NOR's truth table or De Morgan's Law, or simply by negating the Boolean OR oracles, as shown in Fig. 3h and Fig. 3i, respectively. Note that both designs have the same NORing functionality. While the Phase oracle is generated using any design, as demonstrated in Fig. 3g. Both operators (U_s and CU_s) successfully search for solutions for Boolean and Phase oracles, as shown in Fig. 3j–l. Boolean and Phase oracles are generated using the Boolean structure of $\neg(a \vee b)$ based on NOR's truth table or from the Boolean structure of $(\neg a \wedge \neg b)$ based on De Morgan's Laws.

(vi) The Boolean oracle for the classical Boolean XOR gate is designed using either the XOR's truth table or the composite (in-cascade) design, as shown in Fig. 4b and Fig. 4c, respectively. Note that both designs have the same XORing functionality. While the Phase oracle is generated using any design, as demonstrated in Fig. 4a. The CU_s operator successfully searches for all solutions for the Boolean oracle, as illustrated in Fig. 4f, while the U_s operator fails for Phase and Boolean oracles, as demonstrated in Fig. 4d and Fig. 4e, respectively. Hence, the CU_s operator shows its advantage over the U_s operator in searching for solutions for further arbitrary Boolean oracles when their collector gates are in the form of Boolean XOR gates. Note that when searching for solutions for Boolean oracles in the Boolean structure of ESOP using the U_s operator, the implicit XORing functionality of fqubit is utilized, i.e., without using a collector gate in the form of Boolean XOR

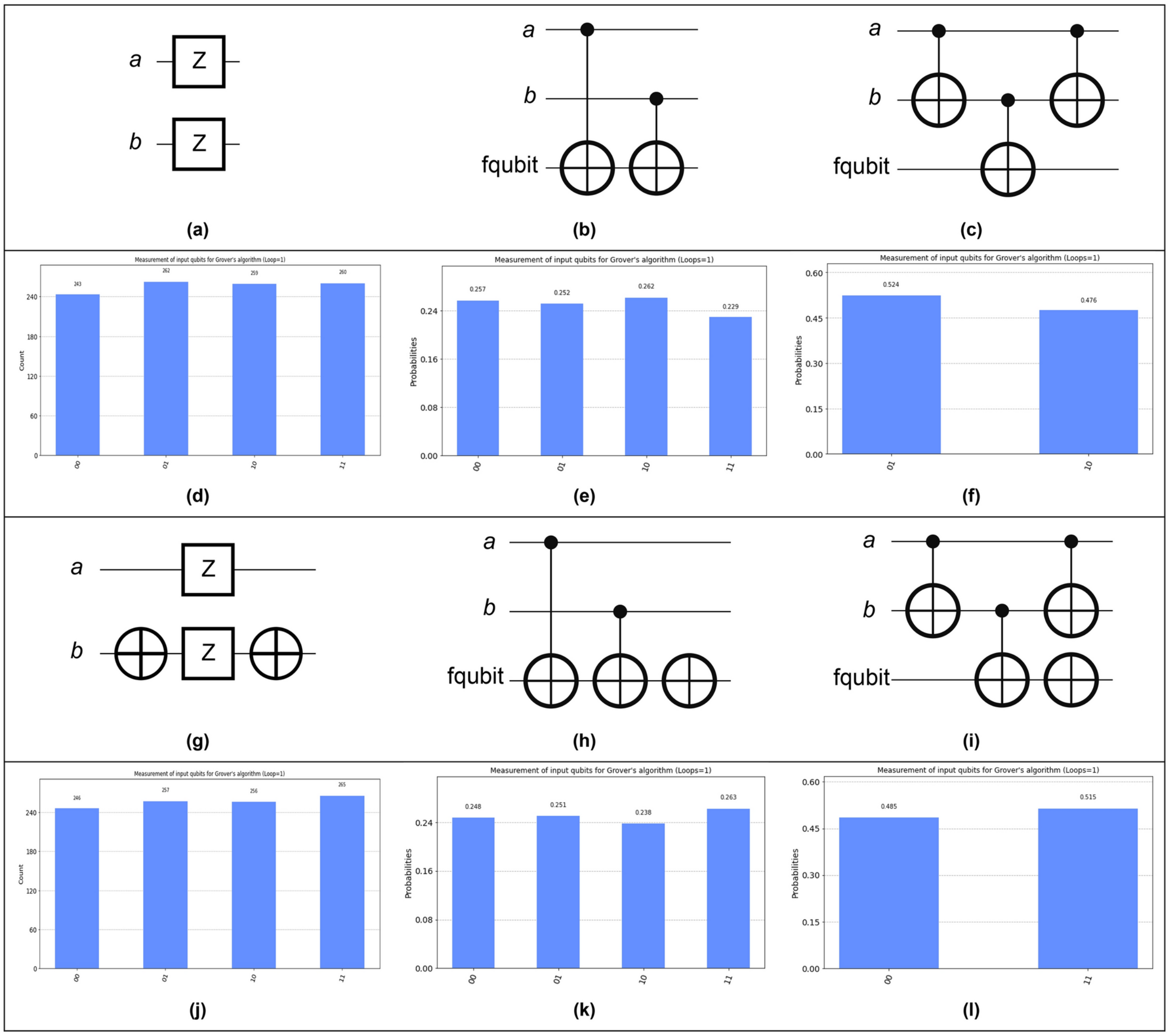
a
b
Z
Z
(a)
a
b
fqubit
(b)
a
b
fqubit
(c)
Measurement of input qubits for Grover's algorithm (Loop=1)
243
262
259
260
Count
00
01
10
11
(d)
Measurement of input qubits for Grover's algorithm (Loops=1)
0.257
0.252
0.262
0.229
Probabilities
(e)
Measurement of input qubits for Grover's algorithm (Loops=1)
0.524
0.476
Probabilities
01
10
(f)
a
b
Z
Z
(g)
a
b
fqubit
(h)
a
b
fqubit
(i)
Measurement of input qubits for Grover's algorithm (Loop=1)
246
257
256
265
Count
(j)
Measurement of input qubits for Grover's algorithm (Loops=1)
0.248
0.251
0.238
0.263
Probabilities
(k)
Measurement of input qubits for Grover's algorithm (Loops=1)
0.485
0.515
Probabilities
00
11
(l)

Fig. 4. Boolean and Phase oracles for classical primitive Boolean gates (XOR and XNOR) with the final measured solutions using the U_s and CU_s operators: (**a**) Phase oracle of classical Boolean XOR gate, (**b**) Boolean oracle of classical Boolean XOR gate from the XOR's truth table, (**c**) Boolean oracle of classical Boolean XOR gate from the composite (in-cascade) design $(a \oplus b \oplus fqubit)$, (**d**) the non-solutions for Phase XOR oracle using the U_s operator, (**e**) the non-solutions for Boolean XOR oracle using the U_s operator, (**f**) the solutions for Boolean XOR oracle using the CU_s operator, (**g**) Phase oracle of classical Boolean XNOR gate, (**h**) Boolean oracle of classical Boolean XNOR gate from the XNOR's truth table, (**i**) Boolean oracle of classical Boolean XNOR gate from the composite (in-cascade) design $\neg(a \oplus b \oplus fqubit)$, (**j**) the non-solutions for Phase XNOR oracle using the U_s operator, (**k**) the non-solutions for Boolean XNOR oracle using the U_s operator, and (**l**) the solutions for Boolean XNOR oracle using the CU_s operator. Note: the CU_s operator successfully searches for all solutions for all Boolean oracles of classical primitive Boolean gates (XOR and XNOR).

gate, as discussed in the following subsection "Oracles for POS, SOP, and ESOP". Boolean and Phase oracles are generated using the Boolean structure of $(a \oplus b)$ based on XOR's truth table or from the Boolean structure of$(a \oplus b \oplus fqubit)$ based on the composite (in-cascade) design.
(vii) The Boolean oracle for the classical Boolean XNOR is designed using either the XNOR's truth table or the composite (in-cascade) design, as shown in Fig. 4h and Fig. 4i, respectively. Note that both designs have the same XNORing functionality. While the Phase oracle is generated using any design, as demonstrated in Fig. 4g. The CU_s operator successfully searches for all solutions for the Boolean oracle, as shown in Fig. 4l, while the U_s operator fails for Phase and Boolean oracles, as shown in Fig. 4j and Fig. 4k, respectively. Therefore, the CU_s operator shows its advantage over the U_s operator in searching for solutions for arbitrary Boolean oracles of collector gates in the form of Boolean XNOR gate. Boolean and Phase oracles are generated using the Boolean structure of $\neg(a \oplus b)$ based on XNOR's truth table or from the Boolean structure of $\neg(a \oplus b \oplus fqubit)$ based on the composite (in-cascade) design.

Concluding that the U_s operator only searches for solutions for Boolean and Phase oracles, when their collector gates are in the forms of Boolean AND gate and Boolean NOR gate! While the CU_s operator successfully searches for all solutions for all Boolean oracles regardless of the Boolean forms of their collector gates.

Oracles for POS, SOP, and ESOP

Three classical Boolean structures of POS, SOP, and ESOP are implemented for arbitrary problems consisting of three classical literals (*a*, *b*, and *c*), and their Boolean and Phase oracles are constructed in various Boolean forms of collector gates. Such that, the POS represents the Boolean structure in (1), the SOP represents the Boolean structure in (2), and the ESOP represents the Boolean structure in (3) below.

$$(a \vee b \vee \neg c) \wedge (\neg a \vee c) \wedge (\neg b \vee c) \tag{1}$$

$$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (\neg b \wedge c) \tag{2}$$

$$(a \wedge b \wedge \neg c) \oplus (\neg a \wedge c) \oplus (\neg b \wedge c) \tag{3}$$

(i) From (1) above, the Boolean POS oracle utilizes the Boolean AND gate as its collector gate, as shown in Fig. 5b, and the Phase POS oracle is generated as shown in Fig. 5a. The CU_s operator successfully searches for all solutions for the Boolean POS oracle, as shown in Fig. 5e. While the U_s operator fails to search for solutions for Phase and Boolean POS oracles, even though the collector gates of these oracles are in the form of Boolean AND gate, as illustrated in Fig. 5c and Fig. 5d, respectively! To the best of our knowledge, so far no one has found that some logical structures of oracles do not work correctly for both Boolean and Phase oracles. This is perhaps for two reasons: (A) most authors formulate constraints satisfaction problems as product of constraints (products of predicates), so that they do not formulate those problems with other collector gates, and (B) the numerical errors in their measurements they attribute perhaps to the errors of computer hardware or models in simulator rather than to the fundamental mistakes in the algebraic design of diffusers!
(ii) From (2) above, the Boolean SOP oracle has its collector gate in the form of Boolean OR gate, as shown in Fig. 5g, and the Phase SOP oracle is generated as shown in Fig. 5f. The CU_s operator successfully searches for all solutions for the Boolean SOP oracle, as presented in Fig. 5(j). While the U_s operator fails for Phase and Boolean SOP oracles, as demonstrated in Fig. 5h and Fig. 5i, respectively. This gives an intuitive thought that the U_s operator always fails in searching for solutions for Boolean and Phase oracles when their collector gates are in the form of Boolean OR gate, as discussed above in the subsection "Oracles of Primitive Boolean Gates (iv)".
(iii) From (3) above, in this article, the Boolean ESOP oracle is implemented to not have any collector gate, since this design relies on the implicit XORing functionality of fqubit, as shown in Fig. 5l. While the Phase ESOP oracle is generated as shown in Fig. 5k. Both operators (U_s and CU_s) successfully search for all solutions for Phase and Boolean oracles, as illustrated in Fig. 5m–o.

For POS and SOP oracles (Boolean and Phase), the U_s operator fails and is not capable of searching for solutions, except for the ESOP oracles (Boolean and Phase). While the CU_s operator successfully searches for all solutions for Boolean oracles in the structures of POS, SOP, and ESOP. We also noticed that the U_s operator fails to search for solutions for (i) POS oracles (Boolean and Phase) even when their collector gates are expressed in the form of Boolean AND gate and (ii) any Boolean oracle that has too many quantum Boolean-based OR gates in its circuit design, based on our experimental observations! Hence, the CU_s operator has the advantage over the U_s operator for searching for all solutions for Boolean oracles regardless of (i) their Boolean structures, (ii) the Boolean forms of their collector gates, and (iii) the number of utilized quantum Boolean-based gates in their circuit design.

Oracles for digital logic circuits

Two digital logic circuits, as arbitrary classical problems, are designed and then transformed into Boolean and Phase oracles, to illustrate how the CU_s operator is a more effective approach than the U_s operator in searching for all solutions for Boolean oracles, when their collector gates are in the form of Boolean OR gate. The first problem is the classical 2-bit inequality comparator, which outputs '1' when both input numbers are unequal, as shown in Fig. 6a. Its equivalent Phase and Boolean oracles are designed as shown in Fig. 6b and Fig. 6c, respectively. The CU_s operator successfully searches for all solutions for the Boolean oracle, as illustrated in

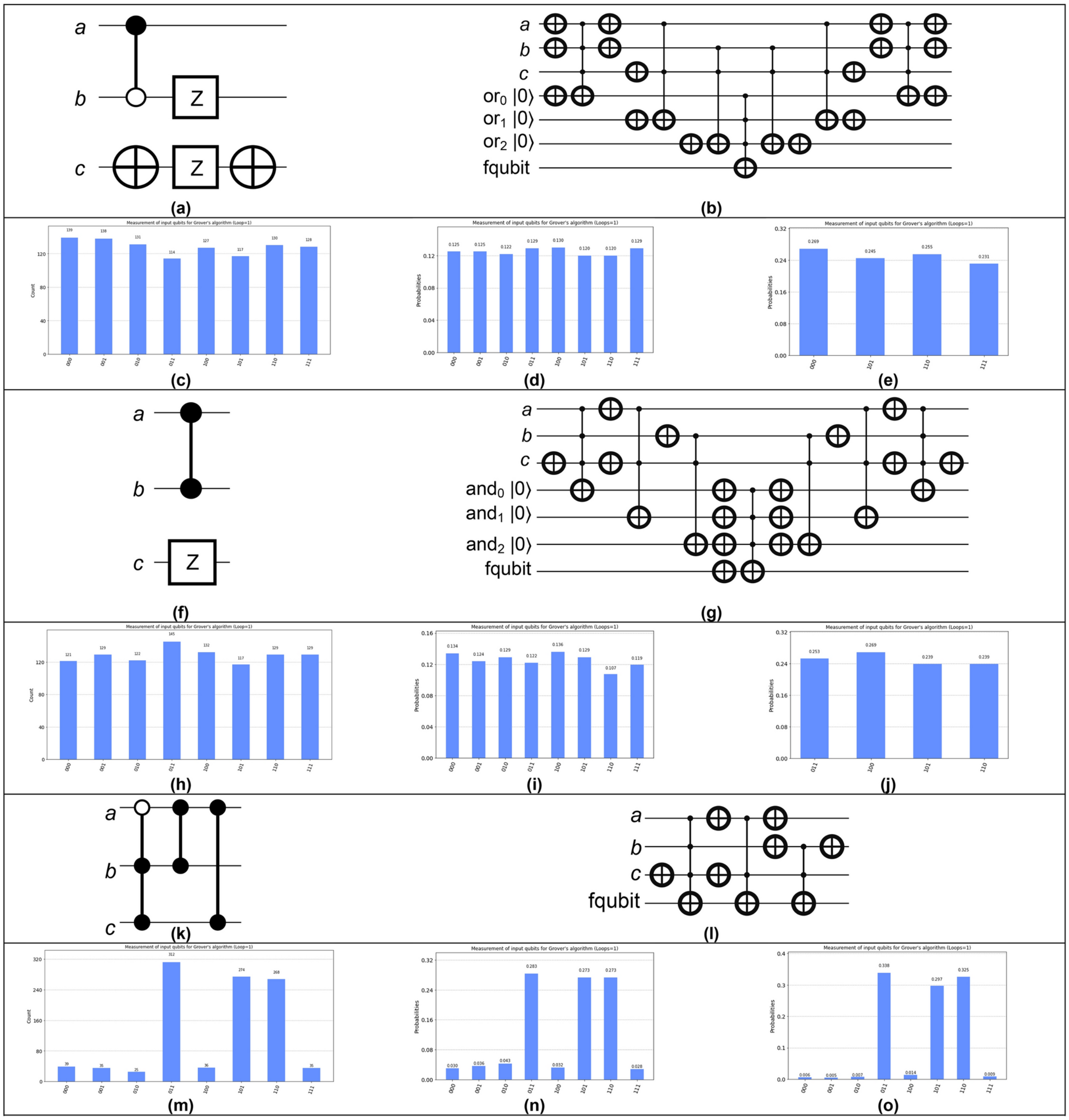

Fig. 5. Boolean and Phase oracles in different Boolean structures (POS, SOP, and ESOP) for arbitrary problems of the literals (a, b, and c): (**a**) Phase POS oracle of $(a \vee b \vee \neg c) \wedge (\neg a \vee c) \wedge (\neg b \vee c)$, (**b**) Boolean POS oracle of $(a \vee b \vee \neg c) \wedge (\neg a \vee c) \wedge (\neg b \vee c)$, (**c**) the non-solutions for the Phase POS oracle using the U_s operator, (**d**) the non-solutions for the Boolean POS oracle using the U_s operator, (**e**) the solutions for the Boolean POS oracle using the CU_s operator, (**f**) Phase SOP oracle of $(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (\neg b \wedge c)$, (**g**) Boolean SOP oracle of $(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (\neg b \wedge c)$, (**h**) the non-solutions for the Phase SOP oracle using the U_s operator, (**i**) the non-solutions for the Boolean SOP oracle using the U_s operator, (**j**) the solutions for the Boolean SOP oracle using the CU_s operator, (**k**) Phase ESOP oracle of $(a \wedge b \wedge \neg c) \oplus (\neg a \wedge c) \oplus (\neg b \wedge c)$, (**l**) Boolean ESOP oracle of $(a \wedge b \wedge \neg c) \oplus (\neg a \wedge c) \oplus (\neg b \wedge c)$, (**m**) the solutions for the Phase ESOP oracle using the U_s operator, (**n**) the solutions for the Boolean ESOP oracle using the U_s operator, and (**o**) the solutions for the Boolean ESOP oracle using the CU_s operator. Note: the CU_s operator successfully searches for all solutions for all Boolean oracles in different Boolean structures (POS, SOP, and ESOP) for arbitrary problems.

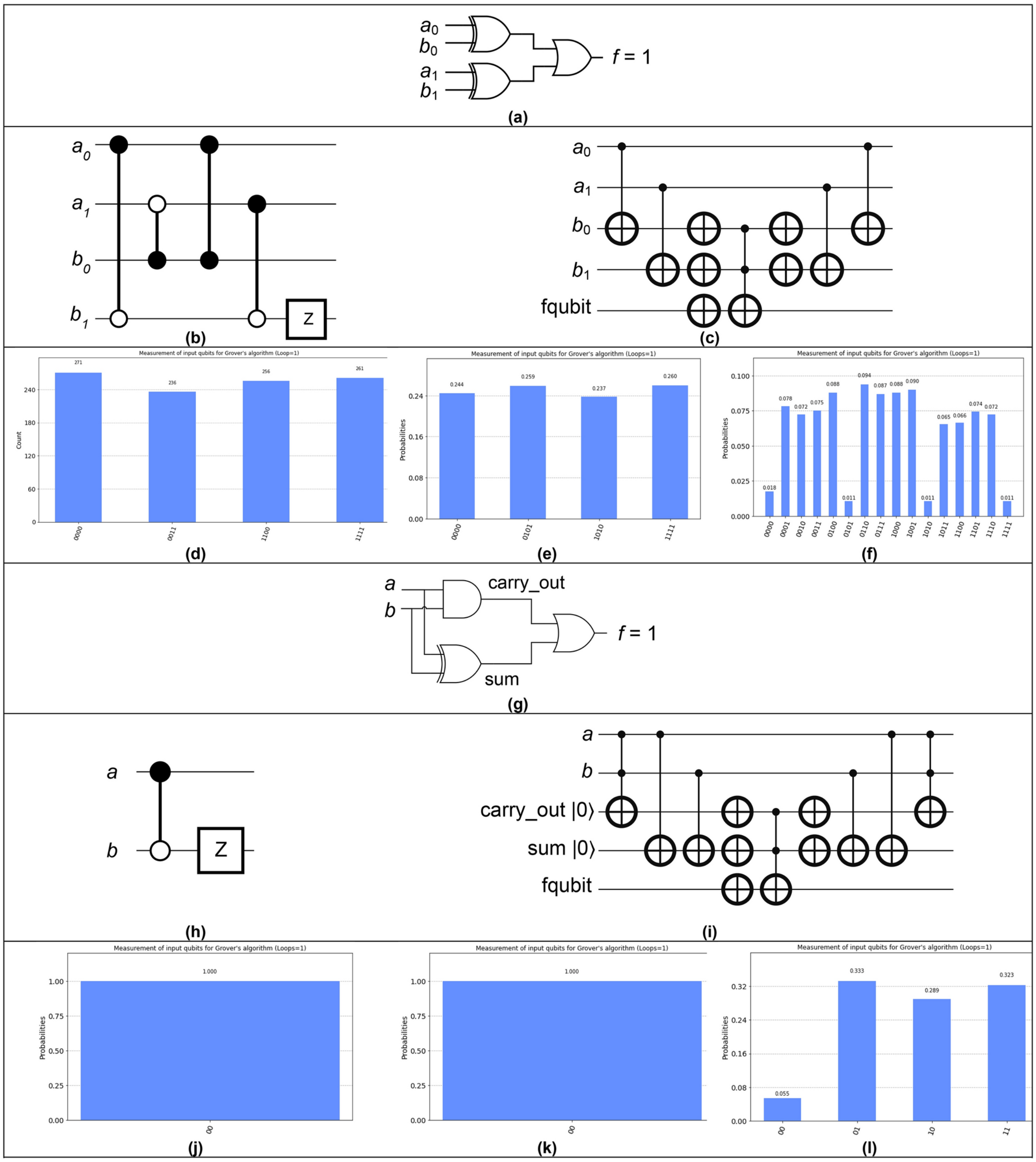

Fig. 6. Phase and Boolean oracles for arbitrary digital logic circuits with the final measured solutions using the U_s and CU_s operators: (**a**) the 2-bit inequality comparator circuit in the classical domain, (**b**) Phase oracle of 2-bit inequality comparator, (**c**) Boolean oracle of 2-bit inequality comparator, (**d**) the partial solutions for (b) using the U_s operator, (**e**) the inverted solutions for (c) using the U_s operator, (**f**) the solutions for (c) using the CU_s operator, (**g**) our designed 1-bit ORed half-adder circuit in the classical domain, (**h**) Phase oracle of 1-bit ORed half-adder, (**i**) Boolean oracle of 1-bit ORed half-adder, (**j**) the non-solution for (h) using the U_s operator, (**k**) the non-solution for (i) using the U_s operator, and (**l**) the solutions for (i) using the CU_s operator. Note: the CU_s operator successfully searches for all solutions for all Boolean oracles of arbitrary digital logic circuits (2-bit inequality comparator and 1-bit ORed half-adder).

Fig. 6f. While the U_s operator fails to search for solutions for the Phase and Boolean oracles, as illustrated in Fig. 6d and Fig. 6e, respectively. Note that the input qubits are denoted in Dirac notation[4,5] of $|\text{MSQ LSQ}\rangle$, where LSQ is the least significant qubit (most-right) and MSQ is the most significant qubit (most-left). Such that, $\text{number}_0 = |a_1 a_0\rangle$ and $\text{number}_1 = |b_1 b_0\rangle$. For this first problem, the Boolean and Phase oracles are generated using the Boolean structure in (4) below.

$$(a_0 \oplus b_0) \vee (a_1 \oplus b_1) \tag{4}$$

The second problem is our designed "1-bit ORed half-adder" that outputs '1' when the sum and/or the carry-out signals have '1', as shown in Fig. 6g, and its equivalent Phase and Boolean oracles are implemented as shown in Fig. 6h and Fig. 6i, respectively. The CU_s operator successfully searches for all solutions for the Boolean oracle, as shown in Fig. 6l, while the U_s operator fails to search for solutions for Phase and Boolean oracles, as illustrated in Fig. 6j and Fig. 6k, respectively. For this second problem, the Boolean and Phase oracles are constructed using the Boolean structure in (5) below.

$$(a \wedge b) \vee (a \oplus b) \tag{5}$$

Note that, for the second problem, the Phase oracle is generated as similar to the Phase OR oracle, as discussed above in the subsection "Oracles of Primitive Boolean Gates (iv)" and shown in Fig. 3a! The U_s operator fails to search for solutions for the two Boolean oracles for the first and second problems, since the U_s operator treats their collector gates in the form of Boolean OR gate as the collector gates in the form of Boolean NOR gate, as also previously stated in the subsection "Oracles of Primitive Boolean Gates (iv)". While the CU_s operator successfully searches for all solutions for these two Boolean oracles, when their collector gates are expressed in the form of Boolean OR gate.

Oracles for CSP-SAT

Two arbitrary classical CSP-SAT problems are designed as Boolean and Phase oracles to search for solutions using Grover's algorithm of both operators (U_s and CU_s). The first problem is the 3-node graph coloring that satisfies, i.e., its output equals to '1', when every two adjacent nodes have different colors, i.e., when every two connected nodes have different values, and these inequality conditions are the so-called "constraints". These constraints for the conditional coloring of these three nodes are mathematically expressed in (6–7) below. Figure 7a presents the classical representation of this problem, and its Phase and Boolean oracles are designed as illustrated in Fig. 7b and Fig. 7c, respectively. Both operators (U_s and CU_s) are employed to search for solutions, and both operators successfully search for all solutions for Boolean and Phase oracles in one Grover iteration, as demonstrated in Fig. 7d–f. Note that (i) each node is represented by one input qubit and (ii) the color of a node is represented by the quantum state of $|0\rangle$ or $|1\rangle$ that met the constraints. For this first CSP-SAT problem, Boolean and Phase oracles are generated using the Boolean structure in (8) below.

$$inequal_0 : node_0 \neq node_1 \tag{6}$$

$$inequal_1 : node_1 \neq node_2 \tag{7}$$

$$(node_0 \oplus node_1) \wedge (node_1 \oplus node_2) \tag{8}$$

The second CSP-SAT problem is the 2×2 Sudoku game that satisfies when all Sudoku's constraints are met. These constraints are: (i) all cells in a row should not have similar values and (ii) all cells in a column should not have similar values, as stated in (9–12) below. Figure 7g demonstrates the classical representation of this problem as the four-cell board layout of Sudoku, and its Phase and Boolean oracles are constructed as shown in Fig. 7h and Fig. 7i, respectively. Both operators (U_s and CU_s) are employed to search for solutions, and both operators successfully search for all solutions for Boolean and Phase oracles, as illustrated in Fig. 7(j–l), in two Grover iterations. Note that (i) each cell is represented by one input qubit and (ii) the value of a cell is represented by the quantum state of $|0\rangle$ or $|1\rangle$ that met the constraints. For this second CSP-SAT problem, Boolean and Phase oracles are generated using the Boolean structure in (13) below.

$$inequal_0 : cell_0 \neq cell_1 \tag{9}$$

$$inequal_1 : cell_0 \neq cell_2 \tag{10}$$

$$inequal_2 : cell_1 \neq cell_3 \tag{11}$$

$$inequal_3 : cell_2 \neq cell_3 \tag{12}$$

$$(cell_0 \oplus cell_1) \wedge (cell_0 \oplus cell_2) \wedge (cell_1 \oplus cell_3) \wedge (cell_2 \oplus cell_3) \tag{13}$$

Consequently, the U_s operator successfully searches for all solutions for Boolean and Phase oracles of the two CSP-SAT problems, since (i) the collector gates for these oracles are in the form of Boolean AND gate and (ii) the quantum Boolean-based OR gates are never used in the circuit design of these oracles. In our experiments, the CU_s operator successfully searches for all solutions for the two CSP-SAT Boolean oracles regardless of their Boolean forms of collector gates as well as the number of utilized quantum Boolean-based gates in their circuit design.

Table 1 states the summary for both operators (U_s and CU_s) in searching for all solutions for the aforementioned arbitrary problems as Boolean and Phase oracles, as well as how many Grover's iterations are required. Note that

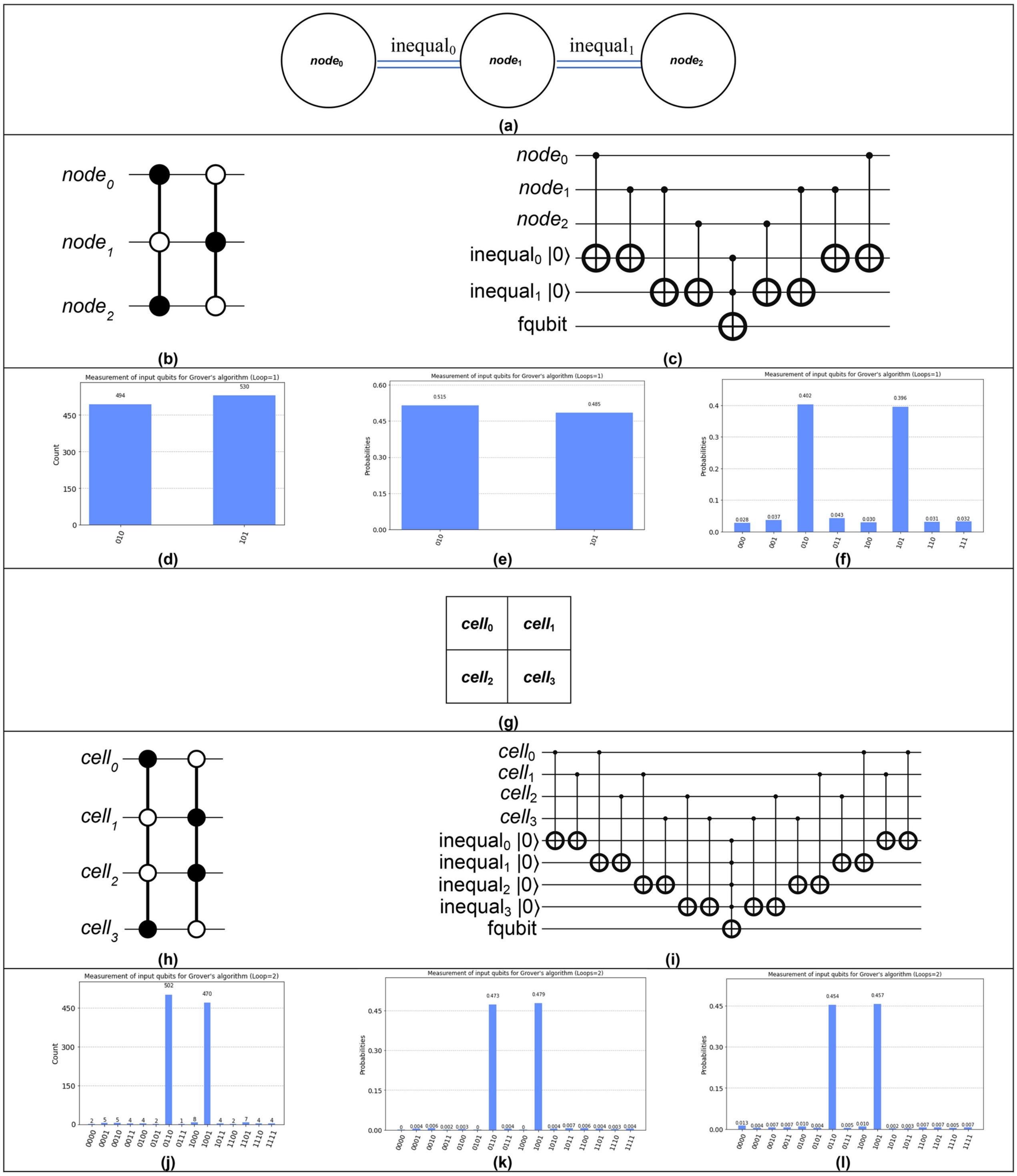
node0
inequal0
node1
inequal1
node2
(a)
(b)
(c)
inequal0 |0⟩
inequal1 |0⟩
fqubit
Measurement of input qubits for Grover's algorithm (Loop=1)
Measurement of input qubits for Grover's algorithm (Loops=1)
(d)
(e)
(f)
cell0
cell1
cell2
cell3
(g)
(h)
(i)
Measurement of input qubits for Grover's algorithm (Loop=2)
Measurement of input qubits for Grover's algorithm (Loops=2)
(j)
(k)
(l)

Fig. 7. Phase and Boolean oracles for arbitrary CSP-SAT problems with the final measured solutions using the U_s and CU_s operators: (**a**) the 3-node graph coloring problem in the classical domain, (**b**) Phase oracle of 3-node graph coloring, (**c**) Boolean oracle of 3-node graph coloring, (**d**) the solutions for (b) using the U_s operator, (**e**) the solutions for (c) using the U_s operator, (**f**) the solutions for (c) using the CU_s operator, (**g**) the 2 × 2 Sudoku board layout in the classical domain, (**h**) Phase oracle of 2 × 2 Sudoku, (**i**) Boolean oracle of 2 × 2 Sudoku, (**j**) the solutions for (h) using the U_s operator, (**k**) the solutions for (i) using the U_s operator, and (**l**) the solutions for (i) using the CU_s operator. Note: both operators (U_s and CU_s) successfully search for all solutions for all Phase and Boolean oracles of CSP-SAT problems.

		Grover's U_s operator				Our CU_s operator	
		Phase oracle		Boolean oracle		Boolean oracle	
Arbitrary problems	Logical structures	Solved?	$O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$	Solved?	$O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$	Solved?	$O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$
$\neg a$	NOT	N.A				True	1
$a \wedge b$	AND	True	1	True	1	True	1
$\neg(a \wedge b)$	NAND	False	1	False	1	True	1
$a \vee b$	OR	False	1	False	1	True	1
$\neg(a \vee b)$	NOR	True	1	True	1	True	1
$a \oplus b$	XOR	None	1	None	1	True	1
$\neg(a \oplus b)$	XNOR	None	1	None	1	True	1
$(a \vee b \vee \neg c) \wedge (\neg a \vee c) \wedge (\neg b \vee c)$	POS (AND-OR)	None	1	None	1	True	1
$(a \wedge b \wedge \neg c) \vee (\neg a \wedge c) \vee (\neg b \wedge c)$	SOP (OR-AND)	None	1	None	1	True	1
$(a \wedge b \wedge \neg c) \oplus (\neg a \wedge c) \oplus (\neg b \wedge c)$	ESOP (XOR-AND)	True	1	True	1	True	1
2-bit inequality comparator: $(a_0 \oplus b_0) \vee (a_1 \oplus b_1)$	DNF-XOR SAT (OR-XOR)	Partial	1	False	1	True	1
1-bit ORed half-adder: $(a \wedge b) \vee (a \oplus b)$	OR-AND/XOR	False	1	False	1	True	1
3-node graph coloring: $(node_0 \oplus node_1) \wedge (node_1 \oplus node_2)$	CNF-XOR SAT (AND-XOR)	True	1	True	1	True	1
2×2 Sudoku game: $(cell_0 \oplus cell_1) \wedge (cell_0 \oplus cell_2) \wedge (cell_1 \oplus cell_3) \wedge (cell_2 \oplus cell_3)$	CNF-XOR SAT (AND-XOR)	True	2	True	2	True	2

Table 1. Summary of Grover's algorithm solving arbitrary problems using the Grover diffusion operator (U_s) and our proposed controlled-diffusion operator (CU_s).

"N.A." means that an operator is not applicable to be designed for Grover's algorithm, "True" means that an operator searches for all solutions, "False" means that an operator searches for all solutions when an oracle is inverted in design, i.e., when inverting a problem, "Partial" means that an operator searches for a few solutions, and "None" means that an operator gives random outputs, i.e., the superposition states of all input qubits for an oracle.

Accordingly, the CU_s operator successfully and effectively searches for all solutions for Boolean oracles regardless of (i) their Boolean structures, (ii) the Boolean forms of their collector gates, and (iii) the number of utilized quantum Boolean-based gates in their circuit design. Grover's algorithm of CU_s operator will provide broad opportunities to solve and search for all solutions for many problems (as Boolean oracles) that are neither designed nor solved using Grover's algorithm of U_s operator. Thus, various problems and applications constructed in different Boolean structures of POS, SOP, ESOP, XOR-SAT, digital logic circuits, digital synthesizers, robotics, and machine learning can be realized as Boolean oracles, and then solved with all k solutions using Grover's algorithm of our proposed CU_s operator, in $O\left(\frac{\pi}{4}\sqrt{\frac{N}{k}}\right)$ Grover iterations for $k \geq 1$.

Discussion

In this article, we noticed that the Grover diffusion operator (U_s) of Grover's algorithm searches for solutions for Boolean and Phase oracles (U_ω), when their collector gates are in the form of Boolean AND gate, i.e., the Toffoli gate for a Boolean oracle and the multiple-controlled Z gate for a Phase oracle. The collector gate is a quantum gate that acts as the junction gate(s) for the Boolean structure of an oracle that represents a problem. In contradiction, when a Boolean or Phase oracle utilizes its collector gate in a different form of Boolean gate, such as NAND, OR, NOR, or XNOR, then the U_s operator fails to search for all solutions! Our work redesigns the U_s operator to a controlled-diffusion operator for Boolean oracles only, which is termed the controlled-U_s (CU_s). The CU_s relies on the computational basis states of the functional, or output, qubit (fqubit), instead of relying on the phase kickback. The CU_s operator is essentially the U_s operator controlled by the states of fqubit. For the CU_s operator, the fqubit is initialized to the state of $|0\rangle$ instead of $|-\rangle$. The collector gate of a Boolean oracle flips the states of fqubit to indicate the marked elements and the unmarked elements for a problem. The U_s operator searches for solutions by rotating and amplifying the amplitudes of marked elements, throughout the effect of phase kickback for a Boolean oracle or the phase inversion for a Phase oracle. While our proposed CU_s operator searches for solutions by rotating and amplifying the amplitudes of marked elements, based on the states of fqubit for a Boolean oracle only. Such that the state of $|0\rangle$ represents an unmarked element, while the state of $|1\rangle$ represents a marked element. In our research, different Boolean and Phase oracles for arbitrary problems in the Boolean structures of POS, SOP, ESOP, and XOR-SAT are designed with various Boolean forms of collector gates, and these oracles are then evaluated using Grover's algorithm of both U_s and CU_s operators. We concluded that the U_s operator searches for solutions for Boolean and Phase oracles, when their collector gates are in the form of Boolean AND gate, and for a few Boolean structures, specifically the ESOP and XOR-SAT! While our CU_s operator successfully searches for all solutions for all Boolean oracles regardless of their Boolean structures and the Boolean forms of collector gates. In conclusion, utilizing the states of fqubit is a more successful and effective search approach than utilizing the phase kickback or phase inversion for Boolean and Phase oracles, respectively. Therefore, the CU_s operator can replace the U_s operator for Grover's algorithm to search for all solutions for Boolean oracles. Further Boolean oracles can be constructed in different Boolean structures with different Boolean forms of their collector gates for practical applications in the topics of digital synthesizers, robotics, machine learning, just name a few, and then Grover's algorithm of our proposed CU_s operator will successfully and effectively solve such applications, in the quantum domain.

Methods

The algorithms, quantum circuits analyses, and mathematical models for the Grover diffusion operator (U_s) and our proposed controlled-diffusion operator (CU_s) are discussed and illustrated in the "Supplementary Information".

Data availability

All relevant data are available from the corresponding author (A.A.-B) upon request.

Received: 29 May 2023; Accepted: 27 September 2024
Published online: 09 October 2024

References

1. Grover, L. K. A fast quantum mechanical algorithm for database search. In *Proc. of the 28th Ann. ACM Symp. on Theory of Computing*. 212–219 (1996).
2. Grover, L. K. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.* **79**(2), 325 (1997).
3. Grover, L. K. A framework for fast quantum mechanical algorithms. In *Proc. of the 30th Ann. ACM Symp. on Theory of Computing*. 53–62 (1998).
4. Nielsen, M. A. & Chuang, I. L. *Quantum Computation and Quantum Information* (Cambridge University Press, 2010).
5. LaPierre, R. *Introduction to Quantum Computing* 1st edn. (Springer, 2021).
6. Boyer, M., Brassard, G., Høyer, P. & Tapp, A. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics* **46**(4–5), 493–505 (1998).
7. Brassard, G., Høyer, P. & Tapp, A. Quantum counting. In *Automata, Languages and Programming* (eds Brassard, G. et al.) (Springer, 1998).
8. Brassard, G., Hoyer, P., Mosca, M. & Tapp, A. Quantum amplitude amplification and estimation. *Contemp Math.* **305**, 53–74 (2002).
9. Figgatt, C. et al. Complete 3-qubit Grover search on a programmable quantum computer. *Nat. Commun.* **8**(1), 1–9 (2017).
10. Li, W. et al. Parameterized algorithms of fundamental NP-hard problems: A survey. *Hum.-Centric Computing Inf. Sci.* **10**(1), 1–24 (2020).
11. Huang, W. & Zhang, L. X. A note on the invariance principle of the product of sums of random variables. *Electronic Commun. Probab.* **12**, 51–56 (2007).
12. Zimmermann, R. & Tran, D. Q. Optimized synthesis of sum-of-products. *IEEE 37th Asilomar Conf. on Signals, Systems & Computers*. 867–872 (2003).
13. Sasao, T. EXMIN2 a simplification algorithm for exclusive-or-sum-of-products expressions for multiple-valued-input two-valued-output functions. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **12**(5), 621–632 (1993).
14. Song, N. & Perkowski, M. Minimization of exclusive sum-of-products expressions for multiple-valued input, incompletely specified functions. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **15**(4), 385–395 (1996).
15. Mishchenko, A. & Perkowski, M. Fast heuristic minimization of exclusive sums-of-products. *Proc. RM'2001 Workshop*. 242–250 (2001).
16. Schaeffer, B., Tran, L., Gronquist, A., Perkowski, M. & Kerntopf, P. Synthesis of reversible circuits based on products of exclusive or sums. In *2013 IEEE 43rd International Symposium on Multiple-Valued Logic* (eds Schaeffer, B. et al.) (IEEE, 2013).
17. Creignou, N. & Daude, H. Satisfiability threshold for random XOR-CNF formulas. *Discret. Appl. Math.* **96**, 41–53 (1999).
18. Ibrahimi, M., Kanoria, Y., Kraning, M. & Montanari, A. The set of solutions of random XORSAT formulae. In *Proc. of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms* (eds Ibrahimi, M. et al.) (SIAM, 2012).
19. Soos, M. & Meel, K. S. 2019 BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting. In *Proc. of the AAAI Conf. on Artificial Intelligence*. **33**(1), 1592–1599 (2019).
20. Hurley, P. J. *A Concise Introduction to Logic* 12th edn. (Cengage Learning, 2015).
21. Kurgalin, S. & Borzunov, S. Implementation of Boolean functions in *Concise Guide to Quantum Computing, Algorithms, Exercises, and Implementations*. 37–43 (Springer, 2021).
22. Hadfield, S. On the representation of Boolean and real functions as Hamiltonians for quantum computing. *ACM Trans. Quantum Computing* **2**(4), 1–21 (2021).
23. Toffano, Z. & Dubois, F. Adapting logic to physics: the quantum-like eigenlogic program. *Entropy* **22**(2), 139 (2020).
24. Schmitt, B. & De Micheli, G. Tweedledum: a compiler companion for quantum computing. In *2022 Design, Automation \& Test in Europe Conference \& Exhibition (DATE)* (eds Schmitt, B. & De Micheli, G.) (IEEE, 2022).
25. Schuler, R. An algorithm for the satisfiability problem of formulas in conjunctive normal form. *J. Algorithms.* **54**(1), 40–44 (2005).
26. Rijnbeek, P. R. & Kors, J. A. Finding a short and accurate decision rule in disjunctive normal form by exhaustive search. *Mach. Learn.* **80**(1), 33–62 (2010).
27. Brown, S. & Vranesic, Z. *Fundamentals of Digital Logic with VHDL Design* 3rd edn. (McGraw Hill, 2008).
28. Perkowski, M. & Liu, K. Binary, multi-valued and quantum board and computer games to teach synthesis of classical and quantum logic circuits. In *2021 IEEE 51st International Symposium on Multiple-Valued Logic (ISMVL)* (eds Perkowski, M. & Liu, K.) (IEEE, 2021).
29. Perkowski, M. Inverse problems, constraint satisfaction, reversible logic, invertible logic and Grover quantum oracles for practical problems. *Sci. Computer Program.* **218**, 02775 (2022).
30. Simonis, H. Sudoku as a constraint problem. In *CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems* (ed. Simonis, H.) (Citeseer, 2005).
31. Lynce, I. & Ouaknine, J. Sudoku as a SAT problem. In *Proc. of the 9th Symp. on Artificial Intelligence and Mathematics (AI&M)*. (2006).
32. Zhang, K. & Korepin, V. E. Depth optimization of quantum search algorithms beyond Grover's algorithm. *Phys. Rev. A.* **101**(3), 032346 (2020).
33. Roy, T., Jiang, L. & Schuster, D. I. Deterministic Grover search with a restricted oracle. *Phys. Rev. Res.* **4**(2), L022013 (2022).

34. Mandviwalla, A., Ohshiro, K. & Ji, B. Implementing Grover's algorithm on the IBM quantum computers. In *2018 IEEE Int. Conf. on Big Data*. 2531–2537 (2018).
35. Liu, H., Li, F. & Fan, Y. Optimizing the quantum circuit for solving Boolean equations based on Grover search algorithm. *Electronics* **11**(15), 2467 (2022).
36. Brailsford, S. C., Potts, C. N. & Smith, B. M. Constraint satisfaction problems: algorithms and applications. *Eur. J. Oper. Res.* **119**(3), 557–581 (1999).
37. Rao, P., Yu, K., Lim, H., Jin, D. & Choi, D. Quantum amplitude estimation algorithms on IBM quantum devices. *Quantum Commun. Quantum Imag. XVIII.* **11507**, 49–60 (2020).

Author contributions

A.A.-B organized the research, collected and analyzed data, designed and programmed the controlled-diffusion operator, and contributed to the theory and the manuscript; and M.P. supervised the research and the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-024-74587-y.

Correspondence and requests for materials should be addressed to A.A.-B.