Eﬃcient Quantum Circuit Synthesis for SAT-Oracle With Limited Ancillary Qubit

Shuai Yang, Wei Zi, Bujiao Wu, Cheng Guo, Jialin Zhang, and Xiaoming Sun

Abstract—One of the main concerns in the era of noisy intermediate-scale quantum (NISQ) computing and fault-tolerant quantum computing is the optimization of circuit implementation for quantum oracles, particularly with limited resources. Synthesizing a satisfiability (SAT) oracle, a crucial component in solving SAT problems, presents a significant challenge. The current state-of-the-art implementation of an m-clause SAT-oracle necessitates 2m – 1 ancillary qubits and a linear number of elementary gates. We develop two eﬃcient and ancilla-adjustable synthesis algorithms to reduce the overall quantum resource usage. Our ﬁrst quantum oracle algorithm achieves quadratic optimization in the number of ancillary qubits with merely eight times increased circuit size. We also show that using only three ancillary qubits with quadratic circuit size expansion is enough. Our second algorithm optimizes the circuit depth of the SAT oracle to O(log m) using m ancillary qubits. By running our algorithms on classical intractable SAT instances featured in SAT competitions, the experiment results show that our required quantum resources align well with our theoretical analysis. Our algorithms highlight the scalability of SAT-oracle-based algorithms in near-term quantum devices, such as Grover’s algorithm.

Index Terms—Circuit synthesis, limited ancillary qubit, satisfiability (SAT) problem, SAT-oracle, space-depth tradeoﬀ.

I. INTRODUCTION

QUANTUM computation has been extensively studied since Feynman ﬁrst proposed it in the 1980s [1]. Several quantum algorithms have been proposed, including Shor’s algorithm and Grover’s algorithm [2], [3], which are superior to the best-classical algorithms. As a result, more and more attention is paid to quantum computation [4], [5]. In these quantum algorithms, quantum oracles are used to evaluate the value of the Boolean function [6]. A Boolean function is a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$. The function of a quantum oracle is transforming $|x⟩|c⟩$ into $|x⟩|c ⊕ f(x)⟩$ [6]. We have to decompose the oracle into elementary gates to implement these quantum algorithms on quantum devices. Since we are in a noisy intermediate-scale quantum (NISQ) era, the number of qubits and the fidelity and decoherence time of the elementary gate is still at a low level [7], [8], [9], [10]. Although this quantum oracle can be implemented theoretically, the number of quantum resources may be unavailable in the NISQ era. Further, in the fault-tolerant quantum computation era, it is a crucial goal to implement large-scale quantum algorithms eﬃciently and accurately. Thus, in both the NISQ and fault-tolerant quantum computation era, it is essential to reduce the quantum oracle circuit’s quantum cost, especially for the depth and width of the quantum circuit, which depicts the time and space required for quantum computing, respectively.

Several algorithms are available for synthesizing a quantum oracle [11], [12], [13], [14]. These algorithms focus on different representations for Boolean functions. However, for the conjunction normal form (CNF) Boolean function, these algorithms require exponential running time to synthesize such an oracle. A CNF Boolean function is an AND of several clauses. Each clause is an OR of variables or their negations. The number of variables is usually denoted by $n$, and the number of clauses is denoted by $m$. The width of a clause is the number of variables used in the clause, and the width of a CNF formula $k$ is deﬁned as the maximum width of the clauses. Notice CNF is a special type of the product of sum. We denote the oracle of a CNF formula as the satisfiability (SAT)-oracle.

The well-known NP-hard problem, the SAT problem, determines whether a CNF is satisfiable [15], [16]. SAT problems arise in several practical application domains, such as gene regulatory networks, model checking, electronic design automation, etc. [17], [18], [19]. In both classical and quantum computation, numerous studies have aimed to solve the SAT problem [20], [21], [22], [23], [24]. These quantum algorithms use the SAT-oracle to evaluate the value of the CNF Boolean function. SAT-oracle can also be used in quantum state preparation [25].

We now deﬁne the quantum circuit synthesis problem for SAT-oracle. Given a CNF formula $f$ over $n$ variables $x = (x_1, x_2, \ldots, x_n)$, the task is to construct a quantum circuit $C$ such that $C |\psi⟩|c⟩ = |\psi⟩|c ⊕ f(\psi)⟩$, for any $\psi ∈ \{0, 1\}^n$. For convenience, denote $n$ variables $m$ clauses $k$-CNF as $CNF_{n,m}^k$. One idea described in [26] and [27] is to store the value of clauses in ancilla qubits and then calculate the AND function with a Toffoli gate.1 However, this algorithm fails when the number of ancilla qubits is limited.

1When we wrote this manuscript, the algorithm used in qiskit is first generating all clauses in the clean ancillary qubits and then merging them up. The new algorithm used in qiskit is a heuristic algorithm, which cannot work when the input $n$ is large. Hence, we still compare with it the old synthesis algorithm.

1937-4151 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of California Santa Barbara. Downloaded on March 12, 2024 at 19:01 (UTC) from IEEE Xplore. Restrictions apply.
Inspired by the construction of multicontrolled Toffoli (MCT) in [6], we design an algorithm to synthesize a general quantum AND (OR) circuit for functions rather than variables to conquer the limitation of ancillary qubits. Based on the general AND circuit for function, we design the width-oriented algorithm to synthesize CNF\textsuperscript{\textcircled{a,m}}. The algorithm costs $\ell$ ancillary qubits and $O((km^{1+o(1)})$ elementary gates. The size of the circuit decreases rapidly with the growth of $\ell$, and the number of ancillary qubits can be reduced to 3. Then, we introduce the depth-oriented algorithm to reduce the depth of the quantum circuit to $O((\log km)$ with $km$ ancillary qubits. When the ancillary qubits are limited, the circuit depth is roughly $O((km^{1+o(1)})/\ell)$. The two algorithms’ running time is $O(km)$. The experimental results show that with a tolerable (a constant ratio) increase in the size of the quantum circuit, the number of ancillary qubits is reduced from $2m - 1$ to $2\sqrt{m}$ using the width-oriented algorithm. The depth-oriented algorithm significantly reduces the circuit depth of the SAT-oracle. We also show the application of our algorithms in SAT problems, quantum state preparation problems, and AND-OR tree oracle problems. We give a resource estimate of solving a meaningful SAT problem using Grover’s and our synthesis algorithms.

II. ALGORITHM AND RESULT

This section will introduce two algorithms to design the SAT-oracle circuit. We start with a general AND problem. We design the general AND circuit similar to the construction in [6]. Then, we use the general AND circuit to design the width-oriented algorithm, which focuses on the size and the number of ancillary qubits in the circuit synthesis. Then, we adjust the inner-most recursion in the width-oriented algorithm and design the depth-oriented algorithm, which reduces the circuit depth rapidly. Finally, we show the performance of our algorithms on random CNF instances.

A. General $p$-AND Circuit

We define such a general AND problem as follows: the input contains $p$ Boolean functions $g_1, g_2, \ldots, g_p$ and the corresponding quantum oracles $O_i |x\rangle = |x\rangle | c \oplus g_i(x)\rangle$. The goal is to construct a quantum circuit $p$-GAND such that $p$-GAND$|x\rangle_n |q\rangle_{\ell} = |x\rangle_n |q\rangle_{\ell} | c \oplus (\bigwedge_{i=1}^{p} g_i(x))\rangle$ with $\ell$ ancillary qubits. Let the AND of the input function be the general $p$-AND function, and here, $p$-GAND is the abbreviation of the general $p$-AND circuit. We can define the general $p$-OR circuit similarly.

The circuit synthesis algorithm for $p$-AND circuit is shown in Algorithm 1. Here, Toffoli($a, b, c$) means the Toffoli gate where the control qubits are $a$ and $b$, and the target qubit is $c$. There is also an example shown in Fig. 1 when $p = 3$.

Lemma 1 (General $p$-AND Circuit): For any natural number $p$, general $p$-AND circuit can be implemented with $2p - 2$ dirty ancillary qubits, $O(p)$ Toffoli gate and 4 calls of each $O_i$.

For convenience, let us introduce some notations

$$q_i^* = \begin{cases} q_i \oplus (q_{i-1} \land g_{i+p+1}(x)), & \text{for } 2p - 2 \geq i \geq p + 2 \\ q_i \oplus (q_{i+p-2} \land g_{i+p}(x)), & \text{for } i = p + 1 \\ q_i \oplus g_i(x), & \text{for } p \geq i \geq 1. \end{cases}$$

Let $q_i^* = q_i \oplus (\bigwedge_{k=1}^{p+1} g_k(x)), \quad i \in \{p + 1, p + 2 \ldots 2p - 2\}$.

Let $q_i$ be the target qubit, $q_i^* = q_i \oplus (\bigwedge_{k=1}^{p} g_k(x))$, and $q_i^* = q_i \oplus (q_p \land g_p(x))$. Let $Q_{a, b} = \otimes_{k=1}^{p} |k\rangle$, $Q_{a, b} = \otimes_{k=1}^{p} |q_k\rangle$, and $Q_{a, b} = \otimes_{k=1}^{p} |q_k\rangle$. The number of dirty ancillary qubits $\ell = 2p - 2$. In this article, we may omit the round operator $\lfloor \cdot \rfloor$ when the context is unambiguous.

Proof: Using the Toffoli gate can easily calculate the AND of the variables. To calculate the general $p$-AND function, we design a new algorithm similar to the MCT gate synthesis algorithm. When the ancillary qubits are clean, we can easily save the value of the functions in the ancillary qubits and then implement a Toffoli gate. We add operations to eliminate the dirty information when the ancillary qubits are dirty. We first divide the $C$ into two subcircuits: 1) merge stage $C_1$ and 2) restore stage $C_2$. The quantum circuit $C_1$ construct contains $2p - 3$ steps. After $C_1$, the result is stored in the target qubit, and then the $C_2$ will restore the change in the input and ancillary qubits. These steps can be divided into three phases: 1) Up phase; 2) Top phase; and 3) Down phase. The Up phase first store the information about $q_{p+1-i}$ and oracle $Q_{p+1-i}$ at qubit $q_{2p-1-i}$. The Top phase then merge the information about $O_1$ and $O_2$. In the Down phase, we merge all the information stored in the $O_i$. Notice that the dirty information is added twice under $F_2$. Finally, in $C_2$, we repeat step 2 to step $2p - 4$ to restore the ancillary qubits.

1) Up Phase: In this phase, we add the information in the ancillary qubits to corresponding qubits, which can help to eliminate unexpected information. There are three parts in step $i \in \{p - 2\}$. The details of each part are shown as follows.

a) In step i.1, as well as step i.3, corresponds to a Toffoli gate. The control qubits are $q_{p+1-i}$ and $q_{2p-1-i}$. When $i = 1$, the target qubit is $q_1$, otherwise the target is $q_{2p-i}$.

b) In step i.2, we call the $O_{p+1-i}$ at $q_{p+1-i}$. Initially, the state is at the $Q_{p+1-i} |q_i\rangle$. After step 1.1, we apply a Toffoli gate to store the dirty information in the ancillary qubits $q_p$ and $q_{2p-2}$. Then, the state transfers to $Q_{1, t} |q_i \oplus (q_p \land q_{2p-2})\rangle$.

Then, we apply an $O_{p+1}$ at the qubit $q_{p+1}$. This step adds the information of the oracle into the qubits. The state after step 1.2 becomes $Q_{1, t} |q_i \oplus (q_p \land q_{2p-2})\rangle$.

Finally, in step 1.3, we use a Toffoli gate to add the information to the target qubit, and some dirty information has been eliminated. The state after step 1 is $Q_{1, p-i} |q_i\rangle$.

Similar to the analysis in the step 1, the state before step i.1 is $Q_{1, p-i} |q_i\rangle$.
Fig. 1. Example of a general 3-AND circuit, where $|q_i'\rangle = |q_i \oplus \land g_i(x)\rangle$.

After step i.3, the state transfers to

$$Q_{1,p-i}Q_{p-i+1,p}Q_{p+1,2p-i-1}Q_{2p-i,2p-2} |q_i'\rangle.$$  

2) Top Phase: In this phase, we implement a circuit that merges two clauses and stores the result in an ancillary qubit. There are seven parts in step $p - 1$. The details of each part are shown as follows.

a) In steps $(p - 1), j, j \in \{1, 3, 5, 7\}$, the operation are the same. Each step corresponds to a Toffoli gate. The control qubits of the Toffoli gate are $q_1$ and $q_2$, and the target qubit is $q_{p+1}$.

b) In steps $(p - 1), 2$, as well as $p, 6$, we call the $O_2$ at $q_2$.

c) In steps $(p - 1), 4$, we call the $O_1$ at $q_1$.

The steps from $(p - 1), 1$ to $(p - 1), 3$ are similar to the step in Up phase. The state after step $(p - 1), 3$ is

$$|q_1\rangle Q_{2,2p-2} |q_i'\rangle.$$  

The step $(p - 1), 4$ just apply an Oracle $O_1$. So the state is $Q_{1,2p-2} |q_i'\rangle$.

The step from $(p - 1), 5$ to $(p - 1), 7$ restore the qubit $2$ and store the $g_1 \land g_2(x)$ without dirty information in qubit $p + 1$. The equation below only focuses on the qubits $q_1, q_2,$ and $q_{p+2}$

$$q_1' \langle q_2' | q_{p+1}$$  

$$\rightarrow |q_1'\rangle |q_2'\rangle |q_{p+1} \oplus ((q_1 \oplus g_1(x)) \land (q_2 \oplus g_2(x)))$$  

$$\rightarrow |q_1'\rangle |q_2\rangle |q_{p+1} \oplus ((q_1 \oplus g_1(x)) \land (q_2 \oplus g_2(x)))$$  

$$\rightarrow |q_1\rangle |q_2\rangle |q_{p+1} \oplus ((q_1 \oplus g_1(x)) \land g_2(x))$$  

$$= |q_1'\rangle |q_2\rangle |q_{p+1} \oplus (g_1(x) \land g_2(x)).$$

3) Down Phase: In this phase, we merge all the clauses in the ancillary qubits. With the help of Up phase, the target qubit stores the value of input CNF. There are 3 parts in step $i \in \{p, p + 2, \ldots, 2p - 3\}$.

a) Step i.1, as well as step i.3, corresponds to a Toffoli gate. The control qubits are $q_{i-p+3}$ and $q_{i+1}$. When $i = 2p - 3$, the target qubit is $q_i$, otherwise the target is $q_{i+2}$.

b) In step i.2, we call the $O_{i-p+3}$ at $q_{i-p+3}$.

Algorithm 1: p-AND Circuit Synthesis Algorithm

input : $p$ oracles $O_i$ for each Boolean function $g_i(x)$.

output: A circuit $C$ such that $Cx) |0\rangle |2p-2 - c\rangle = |x\rangle |0\rangle |2p-2 - c\rangle \oplus \land g_i(x)) \forall x \in \{0, 1\}^p$.

for $j$ in $p$ to $3$:

- Toffoli($j, j + p - 2; j + p - 1$);
- Apply $O_j$ on $q_j$;
- Toffoli($j, j + p - 2; j + p - 1$);

/ / Up phase

- Toffoli($1, 2; p + 1$);
- Apply $O_2$ on $q_2$;
- Toffoli($1, 2; p + 1$);
- Apply $O_1$ on $q_1$;
- Toffoli($1, 2; p + 1$);
- Apply $O_2$ on $q_2$;
- Toffoli($1, 2; p + 1$);

/ / Top phase

for $j$ in $3$ to $p$:

- Toffoli($j, j + p - 2; j + p - 1$);
- Apply $O_j$ on $q_j$;
- Toffoli($j, j + p - 2; j + p - 1$);

/ / Down phase

Repeat the circuit on the first $2p - 2$ qubits to reset the ancillary qubits. / / Restore stage

The analysis is similar to the step $(p - 1), 5$ to $(p - 1), 7$. In each step, we restore a qubit $i - p + 3$ and store the correct information in qubit $q_{i+3}$.

After the $C_1$, the state is

$$|q_1\rangle Q_{2,p}Q_{p+1,2p-2} |q_i \oplus f(x)\rangle$$

We need to repeat the steps from 2 to $2p - 4$ to restore the ancillary qubits.

To make a summary, we use at most $8p - 12$ Toffoli gates and call each $O_i$ at most four times. In the merge stage, each step except step $p$ contains 2 Toffoli gates. The total number of Toffoli gates in the merge stage is $2(2p - 3 - 1) + 4 = 4p - 4$. Similarly, we use $4p - 8$ Toffoli gates in the restore stage. So we use at most $8p - 12$ Toffoli gates in p-AND circuit. In each stage, each oracle $O_i$ is called at most 2 times. So each $O_i$ is called at most four times in the p-AND circuit.
Algorithm 2: Width-Oriented Algorithm

input: A CNF_{n,m}^k instance \( f = \bigwedge_{i=1}^{m} C_i \) and the number of ancillary qubits \( \ell \geq 3 \). W.L.O.G, let \( \ell \) be even.

output: A circuit \( C \) such that
\[
C(x) |0\rangle_\ell |c\rangle = |x\rangle |0\rangle_\ell |c \oplus f(x)\rangle, \forall x \in \{0,1\}^n.
\]

\[ d \leftarrow \log_{1/2} m, s \leftarrow 2m^2, \]

for \( j \) in 1 to \( \ell/2 \):
- **Clause** \( ((j-1)s + 1, js, \text{Ancilla}[j], \quad d - 1) \);
- **MCT** (Ancilla, Target);

for \( j \) in 1 to \( \ell/2 \):
- **Clause** \( ((j-1)s + 1, js, \text{Ancilla}[j], \quad d - 1) \);

**Clause(SId,EId,Target,Depth):**
- \( \text{EId} \leftarrow \min[\text{EId}, m] \);
- \( \text{SId} \leftarrow \min[\text{SId}, m+1] \);
- if \( \text{EId} < \text{SId} \):
  - return;
- if \( (\text{Depth}=0) \):
  - Synthesize clauses on target qubit;
  - return;
- \( s \leftarrow (\text{EId} - \text{SId})/(\ell/2 + 1) \);
- Apply \( (\ell/2 + 1)\text{-AND} \text{ circuit, where } O_i \text{ is synthesized by Clause(SId} + (i - 1)s, \text{SId} + is - 1, \text{Ancilla}[j], \text{Depth} - 1) \);
- return;

B. Width-Oriented Algorithm

By recursively calling the circuit introduced in Section II-A, we design two efficient algorithms to implement SAT-oracles. The input CNF will be divided into several blocks recursively. In the innermost sub-block, numbered 1st level block, we use the general \( k \)-OR circuit to generate a sub-CNF. The circuit to construct \( i \)th level block is used as an oracle in \( (i + 1) \)th level block. We give the pseudo-code of the width-oriented algorithm and depth-oriented algorithm in Algorithms 2 and 3, respectively.

**Theorem 1:** By applying Algorithm 2, any instance of CNF_{n,m}^k can be implemented by an \( O(n(km/n)^{1+\log_2(\ell/2+1)} ) \) size circuit with \( \ell \) ancillary qubits.

**Proof:** In Algorithm 2, we divide the clauses in CNF into \( p = \lceil \ell/2 \rceil + 1 \) sub-blocks recursively until in each block only one clause or less. A single clause can be realized by a \( k \)-controlled Toffoli gate and several \( X \) gates, where \( k \) is the width of the clause. In other words, Size_{0}(CNF_{n,1}^k) = O(k).

Lemma 1 shows that we can construct a circuit to compute the AND(OR) of some subfunction. By recursively using the general \( p\)-AND/OR circuit, we can finally construct the circuit for any given CNF. We need \( O(\ell) \) elementary gates in each recursion and call \( 4p \) sub-blocks oracle.

The Size_{\ell}(CNF_{n,m}^k) = 4p \text{Size}_{\ell}(CNF_{n/m,p}^k) + 4\ell. Notice that Size_{\ell}(CNF_{n,1}^k) \leq \text{Size}_{0}(CNF_{n,1}^k) = O(k). Solving this recursion formula, we have that Size_{\ell}(CNF_{n,m}^k) = O(kn^{1+\log_2(\ell/2+1)}).

C. Depth-Oriented Algorithm

The width-oriented algorithm performs well in the size of the circuit. When the ancillary qubits are clean (the initial state is \( |0\rangle \)), we design the depth-oriented algorithm to further reduce the circuit depth, with a little increase in the circuit size. The framework of the depth-oriented synthesis algorithm is similar to the algorithm described in the width-oriented synthesis algorithm. Different from the width-oriented synthesis algorithm, we divide the ancillary qubits into three registers: 1) \( q_{\text{mem}} \); 2) \( q_{\text{dirty}} \); and 3) \( q_{\text{clean}} \). The size of these 3 registers are \( \lceil (S - 1)\ell/S + 1 \rceil, \lceil \ell/S + 1 \rceil, \lceil \ell/S + 1 \rceil \), where \( S = \max[\lceil k/\log \ell \rceil, 1] \). We run the recursive procedure by using \( q_{\text{dirty}} \) as ancillary qubits. The only difference lies in the innermost recursion of our algorithm.

There are four stages in the innermost recursion: 1) the copy stage; 2) the clause stage; 3) the merge stage; and 4) the
reset stage. We copy the information in the input register to \(q_{\text{mem}}\) in the copy stage. Then, at the clause stage, we generate \(l/(2(S + 1))\) clauses and store the result in the first half qubits of \(q_{\text{clean}}\). We use the rest half qubit to calculate the AND of these clauses in parallel and store the result to the \(q_{\text{dirty}}\). Finally, we restore all the registers by applying the reverse of these stages.

**Theorem 2:** By applying Algorithm 3, any instance of \(\text{CNF}_{n,m}^k\) can be implemented by an \(O(k(mS/\ell)^{1+c} \log \ell)\)-depth circuit, where \(\ell\) is the number of ancillary qubits, \(S = \max(k/\log \ell), 1)\) and \(c = \log \ell/m\).

**Proof:** To further reduce the depth of the circuit, we need to parallelize our construct algorithm. In the width-oriented synthesis algorithm, each sub-block is implemented one by one. We try to use some of the ancillary qubits, which are clean, to parallelize the circuit.

The framework of the depth-oriented synthesis algorithm is similar to the algorithm described in the width-oriented synthesis algorithm. Different from the width-oriented synthesis algorithm, we divide the ancillary qubits into three registers: 1) \(q_{\text{mem}}\); 2) \(q_{\text{dirty}}\), and 3) \(q_{\text{clean}}\). The size of these three registers are \([(S + 1)/S + 1], [\ell/S + 1], [\ell/S + 1]\), where \(S = \max(k/\log \ell), 1)\). We run the recursive procedure by using \(q_{\text{dirty}}\) as ancillary qubits. The only difference lies in the innermost recursion of our algorithm. In parallel, we use all the ancillary qubits to synthesize \((\ell/S + 1)\) clauses.

The innermost recursion of our circuit is shown in Fig. 2. There are four stages in the innermost recursion: 1) copy stage; 2) the merge stage; and 3) the reset stage.

For convenience, let \(CO, CL, ME, RE\) denote the copy stage, clause stage, merge stage, and reset stage, respectively. After these four stages, we synthesize a \(f = \bigwedge_{j=1}^{[\ell/S]} C_j \in \text{CNF}_{n,\ell/S}^k\) to the target qubit. Without loss of generality, let \([\ell/S]\) be even.

In the copy stage \(CO\), we copy the information of input qubits to the \(q_{\text{mem}}\) register

\[
CO |x\rangle |0\rangle |q\rangle |0\rangle |q_1\rangle \rightarrow |x\rangle \left( \bigotimes_j |x_j\rangle^{\otimes t_j} \right) |q\rangle |0\rangle |q_1\rangle
\]

where the number \(t_j\) is determined by the input Boolean function. The depth of copy stage is \(\log_2(max_j t_j) = O(\log \ell)\).

In the clause stage, we copy the information in the \(q_{\text{mem}}\) register and input qubits to synthesize clauses in parallel. The result is stored in the first half of the \(q_{\text{clean}}\) register

\[
CL |x\rangle \left( \bigotimes_j |x_j\rangle^{\otimes t_j} \right) |q\rangle |0\rangle |q_1\rangle \\
\rightarrow |x\rangle \left( \bigotimes_j |x_j\rangle^{\otimes t_j} \right) \left( \bigotimes_{i=1}^{[\ell/S]} |C_{2i-1} \land C_{2i}\rangle \right) |q_1\rangle
\]

We can synthesize \(O(\ell/k)\) terms with \(O(k)\)-depth circuit. The total depth of the clause stage is \(O(k \log \ell)\).

In the merge stage, we merge all the clauses stored in the \(q_{\text{clean}}\) to the target qubit

\[
ME \left( \bigotimes_{i=1}^{[\ell/S]} |C_{2i-1} \land C_{2i}\rangle \right) |q_1\rangle \\
\rightarrow \left( \bigotimes_{i=1}^{[\ell/S]} |C_{2i-1} \land C_{2i}\rangle \right) |q_1 \oplus f(x)\rangle
\]

A Toffoli gate can merge two CNF formulae on a clean ancillary qubit. To merge \(\ell/2(S + 1)\) CNF formulae, the depth of merge stage is \(O(\log(\ell/2(S + 1))) = O(\log \ell)\).

We repeat the Copy and clause stages to reset all the ancillary qubits in the reset stage

\[
RE \left( |x\rangle \left( \bigotimes_j |x_j\rangle^{\otimes t_j} \right) |q\rangle \left( \bigotimes_{i=1}^{[\ell/S]} |C_{2i-1} \land C_{2i}\rangle \right) \right) |q_1 \oplus f(x)\rangle \\
\rightarrow |x\rangle |0\rangle |q\rangle |0\rangle |q_1 \oplus f(x)\rangle
\]

We repeat the first two stages, and the depth of the reset stage is \(O(k \log \ell)\).

The depth of the innermost recursion circuit is \(O(k \log \ell)\). In the inner-most recursion, \(\ell/S\) clauses can be synthesized in parallel. We use \(\text{Depth}_f(f)\) to denote the depth of the circuit obtained by the algorithm described in this section to synthesize \(f\) with \(\ell\) ancillary qubits. \(\text{Depth}_f(\text{CNF}_{n,\ell/S}^k) = O(k \log \ell)\). Combine with the recurrence formula in the previous section: \(\text{Depth}_f(\text{CNF}_{n,m}^k) = [2\ell/(S + 1)] \text{Depth}_f(\text{CNF}_{n,2m(S + 1)/\ell}^k)\). Then we have

\[
\text{Depth}_f(\text{CNF}_{n,m}^k) = O \left( k \log \ell \left( \frac{mS}{\ell} \right)^{1+\log_2(\ell/4)} \right)
\]

The depth of the quantum circuit declined rapidly with the growth of the number of ancillary qubits. We also point out the asymptotically lower bound for synthesizing the CNF formula by counting.

**D. Lower Bound**

We will prove that there exists a \(k\)-CNF with \(m\) clauses which need \(\Omega(mk)\) size of quantum circuits to approximate it with any error \(\varepsilon < \sqrt{2}/2\), as depicted in Theorem 3. Here, a unitary \(U\) approximate a unitary \(V\) with an error \(\varepsilon\) means \(\max_{|\psi\rangle} \| U - V \| \psi \| \leq \varepsilon\), where the notation \(\| \cdot \|_2\) is the 2-norm of the vector. To obtain this lower bound, let us first give a lower bound for the number of different \(\text{CNF}_{n,m}^k\).

**Lemma 2:** There are \(\Omega \left( \left( \frac{\ell}{m} \right) \right)\) different instances for \(\text{CNF}_{n,m}^k\).

**Proof:** Note that a CNF formula \(\phi : \{T, F\}^n \rightarrow \{T, F\}\) can be uniquely represented as a Boolean function \(f_{\phi} : \{0, 1\}^n \rightarrow \{0, 1\}\). With a little abuse of symbols, we use the same symbol to represent the input of CNF formula and the corresponding Boolean functions. Let \(\phi = (v_1 \lor \cdots \lor v_k) \land \cdots \land (v_{(k-1)m+1} \lor \cdots \lor v_{km})\) be a \(k\)-CNF formula, where \(v_i \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}\), then it can be represented as a Boolean function

\[
f_{\phi}(x) = \left( 1 - \prod_{j=1}^{k} v_j \right) \cdots \left( 1 - \prod_{j=1}^{k} \neg v_{(k-1)m+j} \right)
\]
where \( \tilde{v}_i = 0 \) iff \( v_i = T \) and \( \tilde{v}_i = 1 \) iff \( v_i = F \), and the input \( x_i \in \{0, 1\} \) of \( \phi \) is associated with \( x_i \in \{T, F\} \) of \( \phi \). Let \( f(x), g(x) \) be two functions (formulas), we say \( f \equiv g \) if \( f(x_1, \ldots, x_n) = g(x_1, \ldots, x_n) \) for any legal input \( (x_1, \ldots, x_n) \). Let

\[
\mathcal{L} := \{ v_1 \lor \cdots \lor v_k | v_i \in \{ \neg x_1, \ldots, \neg x_n \}, v_i \neq \tilde{v}_j \text{ for } i, j \in \{k\} \}
\]

be the set of all clauses with \( k \)-variables, where \( x_j \in \{T, F\} \), and \( \neg x_j \) are the negations of \( x_j \). Let a set of \( k \)-CNF formulas be

\[
\mathcal{A} := \{ \phi | \phi = l_1 \land \cdots \land l_m, l_i \in \mathcal{L}, l_i \neq \tilde{l}_j \text{ for } i \neq j \in \{m\} \}
\]

We would like to show the size of \( \mathcal{A} \) is size(\( \mathcal{A} \)) \( = \left( \binom{n}{m} \right) \). i.e., any two different formulas \( \phi_1 = l_1 \land \cdots \land l_m, \phi_2 = l'_1 \land \cdots \land l'_m \) such that there exists \( j \in \{m\}, l_j \neq l'_j \), we have \( \phi_1 \neq \phi_2 \). By contradiction, suppose \( \phi_1 \equiv \phi_2 \), then \( f_{\phi_1}(x) \equiv f_{\phi_2}(x) \). Let \( g_{\phi}(x) \) satisfies \( |g| = k \) be the summations of all degree-\( k \) terms of \( f_{\phi}(x) \). By the definition of \( f_{\phi}(x) \) in (8) and the fact that \( \tilde{v}_j = \tilde{v}_j \)

\[
g_{\phi}(x) = -\sum_{j=0}^{m-1} \tilde{v}_{j+1} \cdots \tilde{v}_{j(k-1)}.
\]

Since \( f_{\phi_1}(x) \equiv f_{\phi_2}(x) \), then \( \deg(f_{\phi_1} - f_{\phi_2}) = 0 \), i.e., \( g_{\phi_1}(x) \equiv g_{\phi_2}(x) \). Let \( \tilde{v}_1 \cdots \tilde{v}_k \) be one term of \( g_{\phi}(x) \). For a given input \( x = (x_1, \ldots, x_k) \) such that \( x_i = 1 \) when \( x_i \in \{\tilde{v}_1, \ldots, \tilde{v}_k\} \) and \( x_i = 0 \) otherwise. It is easy to check \( g_{\phi_1}(x) = 1 \) iff \( x_1, \ldots, x_k \) is one term of the function \( g_{\phi}(x) \). Hence, \( \tilde{v}_1 \cdots \tilde{v}_k \) is also a term of \( g_{\phi_2}(x) \). Without loss of generality, each term \( \tilde{l}_j = \tilde{v}_{j+1} \cdots \tilde{v}_{j(k-1)} \in g_{\phi_1}, \tilde{l}_j \in g_{\phi_2} \) at the same time. Therefore, \( \phi_1 \equiv \phi_2 \), contrary to the fact that \( \phi_1, \phi_2 \) are two different formulas in \( \mathcal{A} \).

Theorem 3: There exists a CNF\(^k\)\(_{m}\) any quantum circuits approximating it with error \( \varepsilon < [\sqrt{2}/2] \) needs size \( \Omega(km) \).

Proof: Let \( U \in \mathcal{C}^{2^n \times 2^n} \) be a two qubit gate, and the \( \delta \)-discretization of the \((j, k)\)th element \( U_{jk} \) be \( U_{jk}^\delta = \delta(a/\delta) + i\delta(b/\delta) \), where \( U_{jk} = a + ib \). Then we have \( \|U - U^\delta\|_2 < 2\delta \). There are at most \((2/\delta)^{32}\) different \( \delta \)-discretizations \( U^\delta \) for the infinite continuous \( U \) in the space by its definition.

In the following, we prove that any two different instances in \( \mathcal{A} \) do not share any common \( \delta \)-discretization. Hence, we can use the counting method to give a lower bound of the circuit size.

Let \( A_G, A_H \) be the quantum circuit representations of two different instances in \( \mathcal{A} \). Let \( s \) be the maximum size of all the unitaries related to \( A_G \) and \( A_H \). By the fact that the unitary \( U \in \mathcal{C}^{2^n \times 2^n} \) has a \( s \)-size quantum circuit, the following inequalities:

\[
\|A_G - A_G^\delta\| < 2s\delta \leq \varepsilon
\]

\[
|A_H - A_H^\delta| < 2s\delta \leq \varepsilon
\]

hold when \( \delta = [\varepsilon/2s] \), and \( \varepsilon < [\sqrt{2}/2] \). Combined with the fact that \( |A_G - A_H| = \sqrt{2} \), we have \( A_G^\delta \neq A_H^\delta \).

Hence, any two different instances in \( \mathcal{A} \) have different \( \delta \)-discretization. There are \( \Omega\left(\binom{n}{m}\right) \) different instances for \( k \)-CNF with \( m \)-clause by Lemma 3 in main file. By the fact that the number of different instances is upper bounded by the number of \( \delta \)-discretization of quantum circuits

\[
\binom{n}{m} \leq \left( \frac{2^s}{\delta} \right)^n = \left( \frac{n^k}{m^{km}} \right)^s.
\] (9)

Since \( \binom{n}{m} = \Omega((n/k)^k) \) for any \( k \). Then

\[
\left( \frac{n}{m} \right)^s = \Omega\left( \left( \frac{n^k}{m^{km}} \right)^m \right).
\]

By inequality (9), we have \( s = \Omega(km) \) when \( k = o(n) \). When \( k = cn \) for constant \( c < 1 \), by Stirling’s formula, \( \binom{n}{m} = \Omega(2^{km}) \) for some constant \( a < 1 \). Hence

\[
\left( \frac{n}{m} \right)^s = \Omega\left( \left( \frac{2^{km}}{m} \right)^m \right).
\]

combined with inequality (9) give us \( s = \Omega(km) \) when \( k = cn \) for constant \( c < 1 \). This implies the lower bound also holds for any \( k < n \) for general \( k \)-CNF.

Combining the Theorems 1 and 3, we see that our algorithm is asymptotically optimal when the number of ancillary qubits is \( \Omega((km)^k) \) for any \( \varepsilon > 0 \). The classical running time is the same as the number of calls to the clause function, which is polynomial to the input size.

E. Experiments Result

We use random \( k \)-CNF as the experimental benchmark to test the performance of different algorithms. To sample a random \( k \)-CNF, \( n \) variables, \( m \) random \( k \)-CNFs, \( k \) the number of variables is \( \Omega(n/k)^k \) for any \( \varepsilon > 0 \). The classical running time is the same as the number of calls to the clause function, which is polynomial to the input size.
Fig. 4. Performance of algorithms. Choose the random 4-CNF with 40, 80, 400, and 800 variables to compare the performance of our algorithm and qiskit’s. The result of qiskit contains only one point. (a) and (b) Number of ancillary qubits needs to reach a given size. (c) and (d) Change of the depth of circuits synthesized by the depth-oriented algorithm. Here, (a) and (c) result for middle-scale variables SAT-oracle and (b) and (d) result for large-scale variables SAT-oracle.

400, and 800. The number of clauses $m$ we choose in this manuscript is determined by the number of variables $n$ and the width of CNF $k$. When $k$ is 3 and 4, there are $m = \lceil 4.267n \rceil$ and $m = \lceil 9.931n \rceil$, respectively, which is called SAT phase transition [28]. Our algorithm is suitable for all the input. To evaluate the quantum cost to conquer the most difficult SAT instance, we choose such a specific $m$ in our experiments. To verify the relationship between the number of ancillary qubits $\ell$ and the quantum cost of our two algorithms, we choose several different $\ell$. Some are near the $n$, and others are near $2m-1$. The results of different widths (the number of variables in a clause) seem similar. For convenience, we plot the result of 4-CNF in Fig. 4, which is appropriate to show the performance of our algorithms.

In Fig. 4(a) and (b), we compare the size of the quantum circuit synthesized by our algorithm and the CNF synthesis algorithm used in the qiskit. The circuit synthesis algorithm in qiskit requires $2m-1$ ancillary qubits, so there is only one point and the corresponding horizontal dashed line in Fig. 4. This Figure shows that when $2m-1$ ancillary qubits are used, the width-oriented algorithm can generate a quantum circuit with a smaller size than qiskit. If we want to significantly reduce the number of ancillary qubits, the corresponding size will only increase by a constant multiple. For example, for a random 4-CNF with $n$ equal to 80, qiskit needs 1587 ancillary qubits to synthesize a quantum circuit of size 103205, while the width-oriented algorithm can use 80 ancillary qubits (about 5% of qiskit) to synthesize a circuit of size 391760 (less than four times).

In Fig. 4(c) and (d), a more notable advantage appears in comparing the circuit depth between the depth-oriented algorithm and the algorithm used in qiskit. The depth-oriented
algorithm requires very few ancillary qubits to synthesize low-depth quantum circuits. When using the same $2m - 1$ ancillary qubits as qiskit, the circuit depth synthesized by the depth-oriented algorithm is significantly lower. For example, for a random 4-CNF with $n$ equal to 800, qiskit needs 15887 ancillary qubits to synthesize a quantum circuit of depth 778498. In comparison, the depth-oriented algorithm can use 200 ancillary qubits (about 1.2% of qiskit) to synthesize a circuit of depth 416179 (about 53.5% of qiskit). If the depth-oriented algorithm uses 15887 ancillary qubits, the depth of the circuit can be reduced to 21735 (about 2.8% of qiskit). For both two algorithms, the quantum cost of the output circuit declines with the growth of the number of ancillary qubits. Despite a few single points, our experimental results agree with the theory. We also design comparisons for specific SAT-oracle instances in Section III.

III. APPLICATION

A. Solve SAT Problem

The SAT problem is one of the most well-known computational problems. The SAT problem is defined as follows: given a CNF Boolean function $f(x): \{0, 1\}^n \rightarrow \{0, 1\}$, and the task is to determine whether the function is satisfiable. Here, a function is satisfiable, which means there is an assignment of variables such that $f(x) = 1$. Several quantum algorithms are introduced to solve the SAT problem. Grover’s algorithm is the most natural and straightforward of these quantum algorithms.

Grover’s algorithm is designed to search for the target item in an unstructured database. In Grover’s algorithm, we need to use $O(\sqrt{N})$ oracles, which can calculate the value of the Boolean function.

Further, we apply our synthesis algorithms to estimate the quantum resources required for solving $k$-SAT using Grover’s algorithm. Our algorithms give a method to synthesize the oracle used in Grover’s algorithm.

In [27], they also estimate the quantum resources required for solving the 14-SAT problem. To solve a 65 to 78 variables 14-SAT, the number of ancillary qubits used is $10^{12}$ to $10^{14}$, which is unavailable in NISQ era. Hence, we fixed the number of ancillary qubits as 240, which is more available in the NISQ era. In Grover’s algorithm, there are multiple rounds of Grover iteration. Table I shows the quantum resources needed for the full round (reaching the highest-success probability) and one round execution of Grover iteration. For example, a random 7-SAT with 80 variables, which is among the most challenging instances that a classical computer can solve today [29], [30], [31], [32], [33], can be solved by using 240 ancillary qubits and a $7.3 \times 10^{18}$-size quantum circuit via Grover’s algorithm.

We also selected 5 CNF files from SAT 2023 [34] as benchmarks for conducting numerical simulation experiments. The experiments showed that our two algorithms have advantages over qiskit in terms of circuit implementation. The width-oriented algorithm can reduce the number of ancillary qubits to 2.5% of qiskit’s by increasing the circuit size by only four times. The depth-oriented algorithm can reduce the circuit depth to 1%−10% of the qiskits and use only 10% of the ancillary qubits. It can be found that our algorithms still perform well in the practical instances.

B. Prepare Quantum State

The quantum state preparation problem is defined as follows. For the input vector $v = (v_0, v_1, \ldots, v_{n-1})$, the problem aims to find a quantum circuit $C$, such that $C|0\rangle = |\psi_v\rangle$.

In [25], they give a framework for quantum state preparation using the SAT-oracle. They first apply a layer of single qubit gate to transfer the state $|0\rangle$ to $\bigotimes_{i=0}^{n-1} |v_i\rangle |0\rangle + |\beta_1\rangle |1\rangle$.

Here, $|0, 1\rangle = \bigcup_{i=0}^{n} |0\rangle_i, |1\rangle_i$ and $|\beta_1\rangle = \sqrt{\sum_{x=0}^{2^n-1} v_x^2} |0\rangle$.

Then, they use a DNF formula function to generate the corresponding amplitudes for the computational bases. Define $n$ different function $f_j(x), j \in [n]$, where

$$f_j(x) = \bigvee_{i \in [0, 1]^i} \bigwedge_{1 \leq i \leq j} (x_{ci} = t_i).$$

After applying the circuit $U_f$ that $U_f |x\rangle |0\rangle = |x\rangle |f(x)\rangle$ on the state $\bigotimes_{i=0}^{n-1} |v_i\rangle |0\rangle + |\beta_1\rangle |1\rangle$. The state is $\sum_{x=0}^{2^n-1} v_x |G\rangle |x\rangle$. Here, the state $|G\rangle$ means some extra information that can be restored with a simple circuit.

For different $j$, the values used in $f_j(x)$ are distinct. Thus, we can operate the circuit of $f_j(x)$ in parallel. Actually, we find the DNF can be further optimized to an n-clause n-width DNF. Using the depth-oriented algorithm with $O(n^2)$ ancillary qubits, the circuit depth of which can be reduced to $O(\log n)$, compared to $O(n^2)$ with qiskit.

C. AND-OR Tree

Our algorithm can be used to synthesize a more general Boolean function, which is represented by the AND-OR tree. AND-OR tree is a tree-like data structure. The leaf node stores

| $k$ | $n$ | #clause | #ancilae | size (Alg.1) | depth (Alg.1) | size (Alg.2) | depth (Alg.2) |
|-----|-----|---------|----------|-------------|--------------|-------------|--------------|
| 3   | 40  | 170     | 240      | $1.8 \times 10^{10}$ | $2.1384$ | $6.2 \times 10^3$ | $7523$ |
| 3   | 80  | 341     | 240      | $4.3 \times 10^{16}$ | $49252$ | $1.0 \times 10^{10}$ | $11586$ |
| 5   | 40  | 844     | 240      | $4.3 \times 10^{11}$ | $5.2 \times 10^5$ | $7.6 \times 10^{10}$ | $92444$ |
| 5   | 80  | 1689    | 240      | $1.0 \times 10^{18}$ | $1.2 \times 10^6$ | $1.1 \times 10^{17}$ | $1.3 \times 10^5$ |
| 7   | 40  | 3511    | 240      | $3.4 \times 10^{12}$ | $4.1 \times 10^6$ | $6.7 \times 10^{11}$ | $8.1 \times 10^5$ |
| 7   | 80  | 7023    | 240      | $7.3 \times 10^{18}$ | $8.4 \times 10^6$ | $9.3 \times 10^{17}$ | $1.1 \times 10^6$ |
the value of a variable or the negation of a variable, and the inner node stores the AND/OR of all its children. The value of an AND-OR tree is the value of the root node. Each layer of the tree is denoted by AND or OR, which indicates whether the value of this layer is AND or OR of the children. That is, the AND-OR tree alternately calculates the AND/OR.

These two algorithms can be easily extended to the AND-OR tree. Let the number of leaf nodes of the AND-OR tree be \( m \). Then, the size of the result circuit is \( O(m) \), and the notation omits some polylog term of \( m \). The depth of the result circuit can be optimized to \( O(\log^2 m) \) with sufficient ancillary qubits. We can also build a tradeoff between the circuit size/depth and the number of ancillary qubits. More specifically, for a constant-layer AND-OR tree, where the max degree of each layer is \( m_1, m_2, \ldots, m_l \) and \( \Pi_{i=1}^l m_i \sim m \). Then the previous work takes \( m \) ancillary qubits to get an \( O(m) \)-size circuit. Our algorithm can reduce the ancillary qubits to \( m^{1/k} \) with only a constant factor expansion in circuit size.

IV. CONCLUSION

In this manuscript, we develop two efficient and ancilla-adjustable synthesis algorithms. We construct the SAT-oracle by creating a general p-AND circuit. We then utilize a width-oriented algorithm, which recursively employs the p-AND module within the circuit construction process. Notice that this algorithm can utilize temporarily idle qubits as ancillary qubits, which can fully utilize the quantum resource in the NISQ era. The size of the circuit generated by this algorithm is \( O(n (km/n)^{1+\epsilon}) \), where \( c = o(1) \) is determined by \( \epsilon \). This algorithm achieves quadratic optimization in the number of ancillary qubits with eight times increased circuit size. We also prove a matched lower bound of this problem using the counting method. Further, we propose a depth-oriented algorithm to reduce circuit depth. The depth of the circuit generate by depth-oriented algorithm is \( O(k \log (\log m S/\ell)^{1+\epsilon}) \), where \( S = \max \{k/ \log \ell, 1 \} \) and \( c' = o(1) \) is determined by \( \ell \). We designed several experiments to evaluate the performance of our algorithm. Finally, we enumerate some applications of our algorithms, such as the SAT problem, quantum state preparation problem, and AND-OR tree synthesis. Our algorithms can solve these problems more efficiently.

Some interesting open problems left. Can we use the dirty ancillary qubits to replace the clean ancillary qubits in the synthesis algorithm? Is there some essential difference between the clean and dirty ancillary qubits in a general circuit? Are there some efficient algorithms that can generate size-optimal or depth-optimal circuits for any given CNF instance?

REFERENCES

[1] R. P. Feynman, “Simulating physics with computers,” Int. J. Theor. Phys., vol. 21, no. 6, pp. 467–488, 1982.
[2] P. W. Shor, “Polynomial time algorithms for discrete logarithms and factoring on a quantum computer,” in Proc. 1st Int. Symp. Algorithmic Number Theory, Ithaca, NY, USA, 1994, p. 209.
[3] L. K. Grover, “A fast quantum mechanical algorithm for database search,” in Proc. 28th Annu. ACM Symp. Theory Comput., 1996, pp. 212–219.
[4] S. Beigi and L. Taghavi, “Quantum speedup based on classical decision trees,” Quantum, vol. 4, p. 241, Mar. 2020.
[5] F. Magniez, M. Santha, and M. Szegedy, “Quantum algorithms for the triangle problem,” SIAM J. Comput., vol. 37, no. 2, pp. 413–424, 2007.
[6] M. A. Nielsen and I. Chuang, Quantum Computation and Quantum Information. Cambridge, U.K.: Cambridge Univ. Press, 2002.
[7] J. Preskill, “Quantum computing in the NISQ era and beyond,” Quantum, vol. 2, p. 79, Aug. 2018.
[8] A. Frate et al., “Quantum supremacy using a programmable superconducting processor,” Nature, vol. 574, no. 7779, pp. 505–510, 2019.
[9] O. Di Matteo and M. Mosca, “Parallelizing quantum circuit synthesis,” Quant. Sci. Technol., vol. 1, no. 1, 2016, Art. no. 15003.
[10] S. W. Wang, M. Luo, H. Li, Z. Qu, and X. Wang, “Improved quantum ripple-carry addition circuit,” Sci. China Inf. Sci., vol. 59, Feb. 2016, Art. no. 42046.
[11] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, “Synthesis of reversible logic circuits,” IEEE Trans. Comput-Aided Design Integr. Circuits Syst., vol. 22, no. 6, pp. 710–722, Jun. 2003.
[12] D. M. Miller, D. Maslov, and G. W. Dueck, “A transformation based algorithm for reversible logic synthesis,” in Proc. 40th Annu. Design Autom. Conf., 2003, pp. 318–323.
[13] R. Wille and R. Drechsler, “BDD-based synthesis of reversible logic for large functions,” in Proc. 46th Annu. Design Autom. Conf., 2009, pp. 270–275.
[14] K. Fazel, M. A. Thornton, and J. Rice, “ESOP-based Toffoli gate cascade generation,” in Proc. IEEE Pacific Rim Conf. Commun., Comput. Signal Process., 2007, pp. 206–209.
[15] S. A. Cook, “The complexity of theorem-proving procedures,” in Proc. 3rd Annu. ACM Symp. Theory Comput., 1971, pp. 151–158.
[16] L. A. Levin, “Universal sequential search problems,” Problemy Peredachi Informatsii, vol. 9, no. 3, pp. 115–116, 1973.
[17] E. Corblin, L. Bordeaux, Y. Hamadu, E. Fanchon, and L. Trilling, A SAT-Based Approach to Decipher Gene Regulatory Networks, Integrative Post-Genomics, RIAMS, Lyon, France, 2007.
[18] K. L. McMillan, “Interpolation and SAT-based model checking,” in Proc. Int. Conf. Comput.-Aided Verification, 2003, pp. 1–13.
[19] W. Kanz and D. Stoffel, Reasoning in Boolean Networks: Logic Synthesis and Verification Using Testing Techniques, vol. 9, New York, NY, USA: Springer, 1997.
[20] T. Schoning, “A probabilistic algorithm for k-SAT and constraint satisfaction problems,” in Proc. 40th Annu. Symp. Found. Comput. Sci., 1999, pp. 410–414.
[21] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane, “An improved exponential-time algorithm for k-SAT,” J. ACM, vol. 52, no. 3, pp. 357–364, 2005.
[22] T. D. Hansen, H. Kaplan, O. Zamir, and U. Zwick, “Faster k-sat algorithms using biased-PSZS,” in Proc. 51st Annu. ACM SIGACT Symp. Theory Comput., 2019, pp. 578–589.
[23] V. Dunjko, Y. Ge, and J. I. Cirac, “Computational speedups using small quantum devices,” Phys. Rev. Lett., vol. 121, no. 25, 2018, Art. no. 250501.
[24] A. Leporati and S. Felloni, “Three ‘quantum algorithms’ to solve 3-SAT,” Theor. Comput. Sci., vol. 372, nos. 2–3, pp. 218–241, 2007.
[25] G. Rosenthal, “Query and depth upper bounds for quantum unitaries via grover search,” 2021, arXiv:2111.07992.
[26] Qiskit contributors. “Qiskit: An open-source framework for quantum computing,” 2023. [Online]. Available: http://dx.doi.org/10.5281/zenodo.2573505.
[27] E. Campbell, A. Khurana, and A. Montanaro, “Applying quantum algorithms to constraint satisfaction problems,” Quantum, vol. 3, p. 167, Jul. 2019.
[28] D. Mitchell, B. Selman, and H. Levesque, “Hard and easy distributions of SAT problems,” in Proc. AAAI, vol. 92, 1992, pp. 459–465.
[29] “2013 SAT competition.” 2013. [Online]. Available: http://www.satcompetition.org/2013/.
[30] “2014 SAT competition.” 2014. [Online]. Available: http://www.satcompetition.org/2014/.
[31] “2016 SAT competition.” 2016. [Online]. Available: https://baldur.iti.kit.edu/sat-competition-2016/.
[32] “2017 SAT competition.” 2017. [Online]. Available: https://baldur.iti.kit.edu/sat-competition-2017/.
[33] “2018 SAT competition.” 2018. [Online]. Available: https://satcompetition.github.io/2018/.
[34] “2023 SAT competition.” 2023. [Online]. Available: https://satcompetition.github.io/2023/.
Shuai Yang received the bachelor’s degree in computer science from Sichuan University, Chengdu, China, in 2018. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Sciences, Beijing, advised by Prof. J. Zhang. He is interested in problems with the quantum computation. He currently focuses on the quantum circuit synthesis and quantum circuit optimization.

Wei Zi received the B.S. degree in computer science and technology from the University of Chinese Academy of Sciences, Beijing, China, in 2020. He is currently pursuing the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, and the University of Chinese Academy of Sciences advised by Prof. X. Sun. His research interests lie in the field of quantum circuit optimization and quantum algorithms.

Bujiao Wu received the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2021. She is a Postdoctoral Researcher with Peking University, Beijing. During the Ph.D. degree, she worked as a Postdoctoral Researcher with Peking University and Freie Universität Berlin, Berlin, Germany. Her research interests include quantum algorithm, optimization of quantum circuits, quantum random measurements, randomized benchmarking, quantum machine learning, and verification of the quantum advantages.

Cheng Guo received the Doctoral degree from the University of Technology, Sydney, NSW, Australia, in 2015. He was a Postdoctoral Fellow with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, and also with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing. He currently works with the Experimental School Attached To Haidian Teachers Training College.

Jialin Zhang received the Ph.D. degree in applied mathematics from Tsinghua University, Beijing, China, under the supervision of A. C.-C. Yao in 2010. She is currently a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. She is also a Professor with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing. Prior to ICT and UCAS, she was a Postdoctoral Researcher with the University of Southern California, Los Angeles, CA, USA. Her research interest includes quantum computing, submodular maximization, approximation algorithm, and algorithmic game theory.

Xiaoming Sun received the Ph.D. degree from Tsinghua University, Beijing, China, in 2005. He is a Professor with the Institute of Computing Technology (ICT), Chinese Academy of Sciences, Beijing, China, and serves as the Director of the Laboratory for Quantum Computation and Theoretical Computer Science, ICT. Additionally, he is a Professor with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing. His research interests include quantum computing, computational complexity, approximation algorithms, and combinatorics. Prof. Sun is currently the Director of CCF Technical Committee on Theoretical Computer Science.