Efficient deterministic preparation of quantum states using decision diagrams

Fereshte Mozafari[1,*] Giovanni De Micheli,[1,†] and Yuxiang Yang[2,3,‡]

[1]*Integrated Systems Laboratory, Ecole Polytechnique Federal Lausanne, CH-1015 Lausanne, Switzerland*
[2]*Quantum Information and Computation Initiative, Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong, China*
[3]*Institute for Theoretical Physics, ETH Zürich, 8093 Zürich, Switzerland*

(Received 28 April 2022; revised 10 August 2022; accepted 12 August 2022; published 29 August 2022)

Loading classical data into quantum registers is one of the most important primitives of quantum computing. While the complexity of preparing a generic quantum state is exponential in the number of qubits, in many practical tasks the state to prepare has a certain structure that allows for faster preparation. In this paper, we consider quantum states that can be efficiently represented by (reduced) decision diagrams, a versatile data structure for the representation and analysis of Boolean functions. We design an algorithm that utilizes the structure of decision diagrams to prepare their associated quantum states. Our algorithm has a circuit complexity that is linear in the number of paths in the decision diagram. Numerical experiments show that our algorithm reduces the circuit complexity by up to 31.85% compared to the state-of-the-art algorithm, when preparing generic n-qubit states with n^3 nonzero amplitudes. Additionally, for states with sparse decision diagrams, including the initial state of the quantum Byzantine agreement protocol, our algorithm reduces the number of controlled-NOTs by 86.61–99.9%.

DOI: 10.1103/PhysRevA.106.022617

I. INTRODUCTION

Quantum computers are expected to provide advantages in several fields such as optimization [1], chemistry [2], machine learning [3], and materials science [4]. However, the quantum speedups can be sabotaged if the cost of loading data and initialization is too high for the quantum computer [3]. Therefore, minimizing the cost of *quantum state preparation* (QSP), the process of preparing quantum states from their classical descriptions, is a crucial step of quantum computation [5–7].

QSP algorithms for preparing *general n-qubit quantum states* have cost that grows exponentially fast in n [6–10]. Here the cost is quantified by the number of required controlled-NOT (CNOT) gates, as any quantum circuit can be decomposed into CNOT gates and single-qubit gates and the number of single-qubit gates is upper bounded by twice the number of CNOTs [11]. In this paper, we focus on algorithms that prepare quantum states in a deterministic manner with no or fixed ancillary qubit overhead, instead of approximate algorithms [12–14] or algorithms with n-dependent ancilla size [15–17].

In contrast to general quantum states, in most quantum computational tasks, the states to prepare are from subfamilies of n-qubit states, such as uniform quantum states [18,19], Dicke states [20], and cyclic quantum states [21]. In these examples, all state subfamilies have classical descriptions with symmetric structures, which hints at the possibility of utilizing structured classical descriptions of quantum states to achieve efficient QSP. Here we exploit this possibility and propose a QSP algorithm for quantum states represented by reduced ordered decision diagrams (DDs). DDs are directed acyclic graphs over a set of Boolean variables and a nonempty terminal set with exactly one *root* node [22]. DDs avoid redundancies and lead to a more compact representation of logic functions.

In this paper, we consider the preparation of n-qubit quantum states $|\varphi\rangle = \sum_{s\in S}\alpha_s|s\rangle$, i.e., finding a unitary circuit U that consists of elementary quantum gates such that $U|0\rangle^{\otimes n} = |\varphi\rangle$. Here the *index set* $S \subset \{0,1\}^n$ contains every binary string s such that the amplitude α_s of $|s\rangle$ is nonzero, and $\sum_{s\in S}|\alpha_s|^2 = 1$. Without loss of generality, we assume that basis states in S are sorted in descending order. For two arbitrary n-bit strings s and s', there is a natural order $s \succ s'$ if s is no smaller than s' when both are regarded as binary numbers. In this way, we can order the elements of S as $s_1 \succ s_2 \succ \cdots \succ s_m$ and express the state to prepare as

$$|\varphi\rangle = \sum_{i=1}^{m}\alpha_{s_i}|s_i\rangle. \tag{1}$$

We use DDs to represent the state in Eq. (1), where each basis state $|s_i\rangle$ and each amplitude α_{s_i} are represented by a path and a terminal node, respectively. We propose an efficient algorithm that prepares an arbitrary quantum state given its associated DDs. The cost of our algorithm is $O(kn)$, where k is the number of paths in the DD. Since k is always upper bounded by (and can be much smaller than) m, the number of nonzero amplitudes of the state in the computational basis, our algorithm efficiently prepares any *sparse state* with $m \ll 2^n$. Sparse quantum states have many applications for example in

*fereshte.mozafari@epfl.ch
†https://si2.epfl.ch/~demichel/
‡yuxiang@cs.hku.hk

quantum linear system solvers [23], the quantum Byzantine agreement (QBA) algorithm [24] for large n, and quantum machine learning [3]. Additionally, many problems in classical computing are sparse such as sparse (hyper) graph problems [25]. To solve them using a quantum computer, we need to prepare their associated sparse quantum states. In addition, our algorithm can also efficiently prepare states with sparse decision diagrams ($k \ll 2^n$), even if the states themselves are not sparse [$m = \Omega(2^n)$].

Several algorithms have been proposed for sparse quantum state preparation [26–28] with $O(mn)$ cost. In all of them, the idea is based on preparing basis states one by one by applying several CNOTs and one multiple-controlled single-target gate. De Veras *et al.* [26] use one ancilla qubit to avoid disturbing prepared basis states while working on the others. Compared to [27], their results show that their algorithm performs well when the number of 1 bits in binary bit string representation of each basis state is almost 20%, which is a limitation. Malvetti *et al.* [27] propose an algorithm to prepare sparse isometries which include sparse states as well. Gleinig and Hoefler [28] propose an algorithm that works in the opposite direction, i.e., they try to apply some gates to obtain the $|0\rangle^{\otimes n}$ state from the desired sparse state. They repeat the same procedure in m iterations. In every iteration, they select two basis states and merge them into one by applying several CNOTs and one multiple-controlled single-target gate. Comparing methods in [27,28], they both perform well with small m, and their circuit costs are almost the same. However, the idea in [28] is simpler and its classical runtime, which is $O[nm \log_2(m)]$, is less than that of the algorithm in [27], which is $O[\binom{n}{\log_2(m)} + nm^2]$. Hence, we regard [28] as the state of the art (SOTA) and compare our results to it.

Numerical experiments show that our algorithm outperforms the state of the art [28]. Depending on the sparsity m, our algorithm achieves an up to 31.85% reduction of the CNOT cost. The algorithm works very well for the states with sparse decision diagram representations, and uses up to 99.97% fewer CNOTs. In addition, our algorithm requires only one ancilla qubit, in stark contrast to many existing works [15–17] with ancilla qubits that grow with n.

II. RESULTS

A. Decision diagram representation of quantum states

Our quantum state preparation algorithm works efficiently by making use of a data structure named a DD. Here we give a brief introduction to DDs and how they can be used to represent quantum states.

1. Binary decision tree

A binary decision tree is a rooted, directed, acyclic graph that represents a Boolean function $f = f(x_1, x_2, \dots, x_n)$. It consists of a *root* node, several *internal* nodes, and several *terminal* nodes. The root, usually printed as a square labeled f, features the start of the tree. The terminal nodes are labeled 0 and 1. The internal nodes, labeled $x_1, x_2, \dots, x_n$, represent the variables of f. Two adjacent internal nodes x_1 and x_2 are connected by a solid (dotted) arrow called an edge to represent that the parent node x_1 (i.e., the node above) evaluates to 1 (0), and the node x_2 is called the one-child (zero-child) of x_1. A terminal node t has no children and is labeled 1 or 0 depending on the value of f when its variables are evaluated to the values on the path that contains t. Figure 1(a) shows a binary decision tree for the Boolean function $f = x_1\overline{x_2} + \overline{x_1}x_2 + \overline{x_1}\overline{x_2}$.

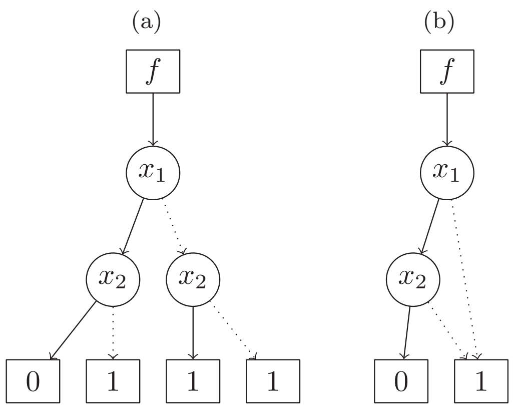

FIG. 1. The decision tree and the decision diagram for $f = x_1\overline{x_2} + \overline{x_1}x_2 + \overline{x_1}\overline{x_2}$. (a) Binary decision tree. (b) Binary decision diagram.

2. Binary decision diagram

A binary decision diagram (BDD) can be obtained from a binary decision tree by applying a *reduction* process, following the rules below.

(1) Two nodes are merged and their incoming edges are redirected to the merged node, if (i) they are both terminal and have the same value or (ii) they are both internal and have the same subgraphs.

(2) An internal node is eliminated, if its two edges point to the same child. After elimination, its incoming edges are redirected to the child.

It is worth mentioning that the reduced tree is also called a *reduced ordered binary decision diagram* (ROBDD) but is commonly referred to as a BDD for simplicity. Figure 1(b) shows the BDD obtained from the decision tree in Fig. 1(a). First, three terminal nodes with value 1 merge to one. Next, node b on the right-hand side of the tree eliminates as both children are terminal node 1.

3. Algebraic decision diagram

An *algebraic decision diagram* (ADD) is the same as a BDD, except that its terminal nodes can have any values [22]. In other words, BDDs are ADDs the terminal nodes of which have binary values. We can still apply reduction rules and get a reduced ordered algebraic decision diagram (ROADD), called an ADD for short.

4. Quantum states represented by DDs

Rather straightforwardly, an arbitrary n-qubit quantum state $|\varphi\rangle = \sum_{s\in S} \alpha_s|s\rangle$ can be represented by a DD: for any $s \in S$, represent s by a path in the tree and set its internal nodes to the qubit registers $q_1, q_2, \dots, q_n$, its edges to solid or dashed lines depending on the state of the registers, and its terminal node to α_s. We then simplify the decision tree by

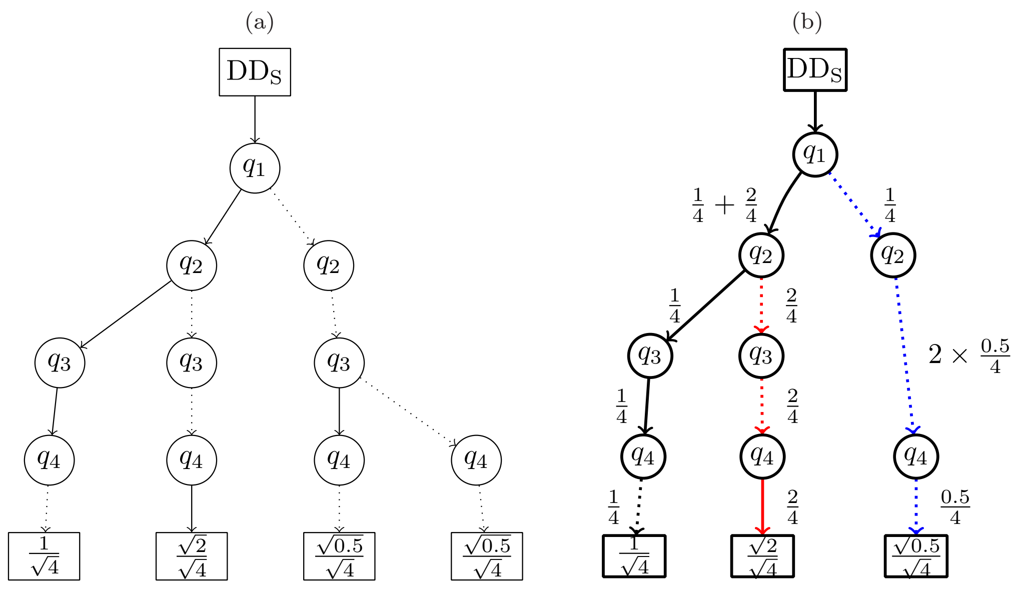

FIG. 2. Decision diagram representation of the quantum state in Example 1. (a) Before applying reduction rules. (b) After applying reduction rules.

removing all the paths corresponding to $s \notin S$ and terminal nodes the values of which are zero. Next, we further apply the reduction rules to get a ROADD (called an ADD for short). When the state is uniform, i.e., all the amplitudes are equal, the ADD can be simplified to a BDD, where a terminal node with the binary value 1 indicates that the associated paths have nonzero amplitudes. Each path p of the reduced DD corresponds to one or more basis states $s \in S_p$, which is a subset of S. Denoting by P the set of the paths of the reduced DD, the state to prepare can be recast in the form

$$|\varphi\rangle = \sum_{p\in P}\sum_{s\in S_p}\alpha_s|s\rangle. \tag{2}$$

Notice that all basis states $s \in S_p$ have the same amplitude.

Example 1. The four-qubit state

$$|\varphi\rangle = \frac{1}{\sqrt{4}}(|1110\rangle + \sqrt{2}|1001\rangle + \sqrt{0.5}|0010\rangle + \sqrt{0.5}|0000\rangle) \tag{3}$$

has index set $S = \{1110, 1001, 0010, 0000\}$ and nonzero amplitudes $\{\frac{1}{\sqrt{4}}, \frac{\sqrt{2}}{\sqrt{4}}, \frac{\sqrt{0.5}}{\sqrt{4}}, \frac{\sqrt{0.5}}{\sqrt{4}}\}$. It can be represented by the DD in Fig. 2(a). We represent each $s \in S$ with a binary string of qubits $q_1q_2q_3q_4$ where q_1, q_2, q_3, and q_4 are internal nodes. Each path shows a basis state s, and the terminal node connecting to each path shows its corresponding amplitude. For example, $\{s_1 = 1110, \alpha = \frac{1}{\sqrt{4}}\}$ expresses that we have a path in which $\{q_1 = 1, q_2 = 1, q_3 = 1, q_4 = 0\}$ that connects to the terminal node $\frac{1}{\sqrt{4}}$. Further notice that on the right-hand side of the diagram [Fig. 2(a)] two terminal nodes are equal, which results in merging them. Furthermore, both left and right subgraphs of q_3 are equal, so this node can be eliminated. Therefore, the DD can be reduced to the ADD in Fig. 2(b) which contains three paths instead of four. Actually, the last two basis states $s_3 = 0010$ and $s_4 = 0000$ correspond to the same path $\{q_1 = 0, q_2 = 0, q_4 = 0\}$.

B. DD-based algorithm for quantum state preparation

In this section, we present our DD-based algorithm for quantum state preparation. We assume that the quantum state to prepare is represented by either an ADD or a BDD (when it is uniform). Using DDs helps us to already have a quantum state without redundancies, which reduces the circuit cost.

Our algorithm works by preparing the paths in a DD one by one. For any n-qubit quantum state to prepare, our algorithm uses only one additional qubit q_A as an ancilla, the value of which is tagged $|\text{yes}\rangle$ (regarded as $|0\rangle$ when used as a control qubit) or $|\text{no}\rangle$ (regarded as $|1\rangle$ when used as a control qubit). Intuitively, q_A serves as an indicator for whether a path has been created in the course of the state preparation. Paths that have been created are marked by $q_A \mapsto |\text{yes}\rangle$ and, by using q_A as control, we can avoid disturbing the created paths when creating a new path.

Each target qubit in our quantum state preparation transforms $|0\rangle$ to a superposition of $\alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2$ ($|\beta|^2$) shows the probability of being zero (one) after measurement. To achieve this transformation, for some nodes, we need to apply a gate, called G, which is explained later. Therefore, we traverse the DD twice: (1) to compute the G gate for each node and (2) to prepare the quantum state.

1. Postorder traversal to compute G gates

We traverse DD in postorder traversal (i.e., visiting one-child, zero-child, and parent nodes). For each node, we compute the probability of being one or zero from its corresponding one-child and zero-child. To compute zero probability (called p_0), for each node, we compute its portion from the one-child (called t_1) and zero-child (called t_0) and then it equals to

$$p_0 = \frac{t_0}{t_1 + t_0}. \tag{4}$$

Algorithm 1 Deterministic Preparation of Quantum States using DD.

```
Input: DD representation of an n-qubit quantum state
       $|\varphi\rangle = \sum_{i=1}^{m} \alpha_{s_i}|s_i\rangle$, and $p_0$ values corresponding to
       each node of DD.
Output: The quantum circuit $qc$ that prepares the desired
        quantum state.
1  Create a quantum circuit $qc$ with $n+1$ qubits corresponding
   to $q_A q_1 q_2 ... q_n$.
2  Initialize the $qc$ with $|1\rangle_{q_A} \otimes |0\rangle_{q_1} |0\rangle_{q_2} ... |0\rangle_{q_n}$.
3  Initiate the pointer current_node as the root of DD.
4  PreOrder_traversal (current_node, qc, p_0_values) :
5  if current_node is a terminal node then
6      Append to qc a multiple-controlled NOT gate with the
       qubits of branches being controls and $q_A$ being the
       target.
7      return
8  if current_node is a branching node then
9      Append to qc a 2-controlled gate $G(p_0)$ [cf. Eq. (5),
       with $p_0$ from $p_0$_values corresponding to
       current_node] gate targeting the qubit corresponding
       to current_node controlled on ancilla qubit and the
       qubit corresponding to the last $|1\rangle$ in the path.
10     Append the qubit corresponding to current_node and
       its value to branches.
11 else
12     if one_child(current_node) $\neq$ nullptr then
13         Append to qc a 2-controlled NOT gate targeting the
           qubit corresponding to current_node controlled on
           ancilla qubit and the qubit corresponding to the
           last $|1\rangle$ in the path.
14 if Some qubits are reduced between current_node and
   one_child(current_node) then
15     Append to qc 2-controlled G($\frac{1}{2}$) gates targeting the
       reduced qubits with ancilla qubit and the qubit
       corresponding to the last $|1\rangle$ in the path as controls.
16 if one_child(current_node) is a terminal node then
17     Append to qc a 2-controlled phase gate that adds a
       phase $e^{i\alpha}$ (corresponding to this path) targeting the
       qubit corresponding to current_node controlled on
       ancilla qubit and the qubit corresponding to the last
       $|1\rangle$ in the path.
18 PreOrder_traversal
   (one_child(current_node), qc, p_0_values)
19 if Some qubits are reduced between current_node and
   zero_child(current_node) then
20     Append to qc 2-controlled G($\frac{1}{2}$) gates targeting the
       reduced qubits with ancilla qubit and the qubit
       corresponding to the last $|1\rangle$ in the path as controls.
21 if zero_child(current_node) is a terminal node then
22     Append to qc a 2-controlled phase gate that adds a
       phase $e^{i\alpha}$ (corresponding to this path) targeting the
       qubit corresponding to current_node controlled on
       ancilla qubit and the qubit corresponding to the last
       $|1\rangle$ in the path.
23 PreOrder_traversal
   (zero_child(current_node), qc, p_0_values)
```

As an example, consider the state in Fig. 2(b); postorder traversal results in first visiting q_4 in the left-hand side. The portion from the one-child is zero and from the zero-child is $|\frac{1}{\sqrt{4}}|^2$ (as it is amplitude we need to square it). Hence, the probability of being zero equals to $\frac{1/4}{0+1/4} = 1$. Next, we go through the upper node q_3; the portion from the one-child comes from the summation of one-child and zero-child portions of q_4, which is $0 + \frac{1}{4}$. The portion from the zero-child is zero and the zero probability is $\frac{0}{1/4+0} = 0$. By continuing this procedure we obtain t_1 and t_0, which are written in the figure on the edges. Note that we need to consider the effect of eliminated nodes. If e nodes are eliminated along an edge, the portion is multiplied by 2^e. For example, in the right-hand side of Fig. 2(b), on the zero-child of q_2, one node (q_3) is removed, which results in $t_0 = 2^1 \times \frac{0.5}{4}$. Finally, α and β for G gates are computed by $\sqrt{p_0}$ and $\sqrt{1-p_0}$, respectively, which we show as

$$G(p_0)|0\rangle = \sqrt{p_0}|0\rangle + \sqrt{1-p_0}|1\rangle. \tag{5}$$

The above $G(p_0)$ can be implemented as a Pauli-y rotation: $G(p_0) = R_y[2\cos^{-1}(\sqrt{p_0})]$.

2. *Preorder traversal to prepare the quantum state*

The algorithm begins with an empty quantum circuit and all qubits initiated as

$$|no\rangle_{q_A} \otimes |0\rangle_{q_1} |0\rangle_{q_2} ... |0\rangle_{q_n}. \tag{6}$$

Starting from the root, the algorithm traverses the DD with preorder traversal (i.e., visiting parent, one-child, and zero-child nodes). To accomplish the traversal, we need to define a pointer *current_node* that points to the current node we are working on. To navigate through the DD, we define functions *one_child* and *zero_child* which return the child of the current node regarding solid and dotted edges, respectively. While traversing through the DD, we compile the state preparation circuit according to the following rules.

(1) *Preparation*: If the current node q is an internal node that is already on a path p_i, we do as follows.

(a) If q is a *branching node*, which means it has both a zero-child and a one-child, we apply to the quantum circuit a 2-controlled $G(p_0)$ gate [see Eq. (5)] on q with q_A and the last node on the path that has a one-child as control qubits, where the value of p_0 is determined by the postorder traversal. Otherwise, q either has a one-child or a zero-child. For the former case, we add a 2-controlled NOT gate on q with q_A and the last node on the path that has a one-child as control qubits. For the latter case, we do nothing.

(b) In addition, we need to consider the effect of reduced nodes between node q and its children. A node is reduced when both its one-child and zero-child point to the same thing. Hence, the qubit with half probability is zero and with half probability is 1. If this is the case, we append to the quantum circuit 2-controlled $G(\frac{1}{2})$ gates on reduced nodes with q_A and the last node on the path that has a one-child as control qubits.

(c) If q is the parent of the ith terminal node, then we add a 2-controlled phase gate on q with q_A and the last node on the path that has a one-child as control qubits, which adds a phase $e^{i\arg(\alpha_i)}$ to the path state $|s_i\rangle$.

(2) *Computing the ancilla*: If the current node is a terminal node, it means that we have prepared the current path. Hence, we need to compute the ancilla qubit to mark that the current path is prepared. We append to the quantum circuit a multiple-controlled-NOT gate on q_A with all qubits at branching nodes on path p_i being control.

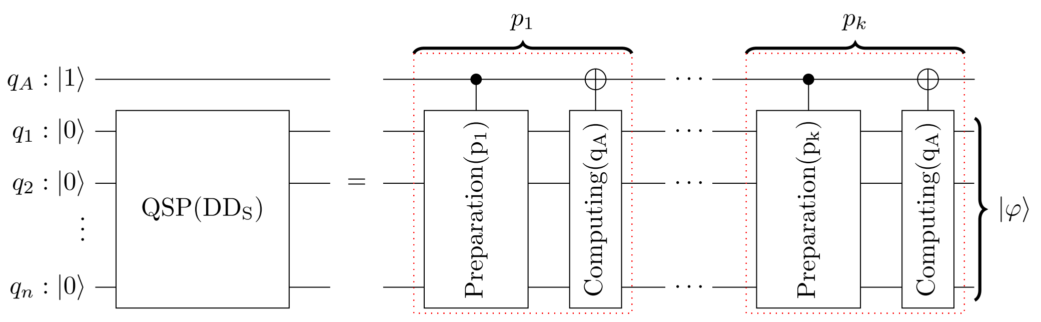

FIG. 3. The general structure of the quantum circuit for QSP over DDs.

This is a recursive traversal where we visit the current node, one-child, and zero-child, respectively. In other words, we prepare paths from the largest (p_1) to the smallest (p_k). In this way, we can order the elements of P as $p_1 \succ p_2 \succ \cdots \succ p_k$. As a result, using this traversal we can prepare basis states in S from the largest (s_1) to the smallest (s_m). The pseudocode of the proposed algorithm is shown in Algorithm 1. Note that, in the postorder traversal, we have already computed p_0 values of G gates corresponding to each node and here we pass it as an argument to the algorithm. Line 5 of Algorithm 1 shows the application of rule 2 (computing the ancilla), and lines 8, 11, 14, 16, 19, and 21 illustrate different conditions of rule 1 (preparation). Additionally, we recursively visit the one-child and zero-child in lines 18 and 23.

Figure 3 shows the general structure of the output quantum circuit of our algorithm. Note that for preparing p_1 the ancilla qubit is not needed, because there is no other path prepared before p_1. Moreover, as p_k is the last path to prepare, we do not need to compute the ancilla qubit.

Example 2. In this example, we show how to create a quantum circuit to prepare the state represented in Fig. 2(b). Preorder traversal helps us to go through three paths presented by black, red, and blue colors. To compute p_0, values of t_0 and t_1 are shown in the figure. Starting from the root, we need to append a $G(\frac{1}{4})$ gate on q_1 that shows the probability of being zero for this qubit. Going through the black path (p_1), on the next node q_2 there exists a branch which requires a 1-controlled $G(\frac{2}{3})$ gate. This is the first basis state and we do not need to check the ancilla qubit. Next, on q_3 there is not any branch but it has a one-child, so it is required to append a CNOT gate with the last $|1\rangle$ in the path (q_2) as control. Next, for q_4 there is not any branch and there is only a zero-child that does not require any action. To compute the ancilla qubit, we need to add a multiple-controlled NOT gate on the ancilla qubit with two controls on branching nodes which are $q_1 = 1$ and $q_2 = 1$.

Afterward, the traversal returns to q_2 and goes through the red path (p_2). It goes to q_3, there is not any branch, and there is only a zero-child that does not require any action. Next, q_4 has a one-child and so we need to add a 2-controlled NOT gate on q_4 with ancilla and q_1 which is the last $|1\rangle$ in the path as control qubits. Then, to mark that p_2 is prepared, we add a 2-controlled NOT gate on the ancilla qubit with $q_1 = 1$ and $q_2 = 0$.

Finally, the algorithm goes back to the root again and traverses the blue path (p_3). q_2 has a zero-child and we do not need to add any gate for it. Next, the q_3 is removed which requires adding a $G(\frac{1}{2})$ gate that shows with the half probability it is zero. There is not any last $|1\rangle$ in this path so it only has one control which is the ancilla. Then, q_4 has a zero-child and again we do not need to add any gate for it. Note that reduced node q_3 here helps us to prepare s_3 and s_4 together. This reduces the number of iterations and so circuit cost. Moreover, as this path corresponds to the last basis states s_3 and s_4, we do not need to compute the ancilla qubit. Figure 4 shows the generated quantum circuit.

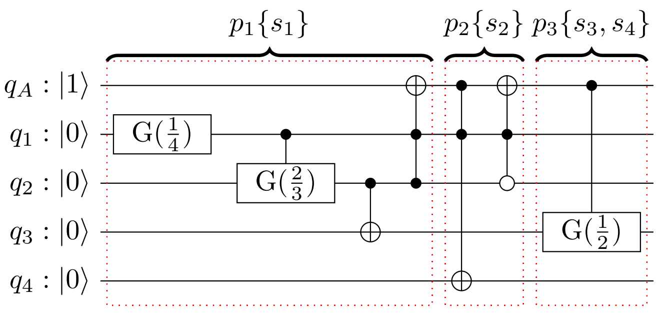

FIG. 4. The generated quantum circuit for preparing the state presented as DD in Fig. 2(b).

C. Numerical experiments

In this section, we evaluate the proposed algorithm over the state of the art [28]. Our algorithm is implemented in an open-source tool, called ANGEL.[1] All experiments are conducted on an Intel Core i7, 2.7 GHz with 16-GB memory.

1. Random states

We evaluate our algorithm on randomly generated states with different amplitudes. The parameter m denotes the number of basis states with nonzero amplitudes. We change m depending on n with different degrees. We compare the size of the circuits produced by our proposed method (PM) with the SOTA method presented in [28]. The final circuits consist of CNOTs and single-qubit gates as elementary quantum gates. We only consider the number of CNOTs as they are more expensive than single-qubit gates in the noisy intermediate-scale

[1]A c++ library for quantum state preparation [29].

TABLE I. Experimental results for quantum states (QS) that have a sparse DD.

			PM				SOTA	
QS	n	m	No. of nodes	No. of reduced nodes	No. of paths k	No. of CNOTs	No. of CNOTs	Imp. (%)
Set 1	20	n	33	6	2	13	275	95.27
Set 2	20	$10n$	41	25	5	190	9983	98.10
Set 3	30	n	60	10	4	62	463	86.61
Set 4	30	$10n$	78	41	9	568	17019	96.66
QBA	20	n^3	32	110	18	1165	1361456	99.91
QBA	25	n^3	37	123	19	1321	2974248	99.95
QBA	30	n^3	44	141	22	1591	5512726	99.97

quantum. But consider that reducing CNOTs means we are reducing single-qubit gates as well. Figure 5 shows results for $n = 16$, 20, 24, and 28. For each combination of parameters shown in the figure, we sampled ten random states and show the average values. Each subfigure shows how the number of CNOTs grows as we increase m as a function of n. For small m, SOTA is better as it is an efficient idea for sparse states. But by increasing m our results close to SOTA and finally for $m = n^3$ PM outperforms SOTA up to 31.85, 17.4, 13.1, and 11.4% for n equal to 16, 20, 25, and 28, respectively. The reason is that in the DD representation, for large m, there is a better sharing between basis states which results in a sparse decision diagram. The results for $n = 16$ are better than those for larger values of n because the percentage of nonzero amplitudes is higher for $n = 16$. Considering the sparsity condition in [28], $m \in o(\frac{2^n}{n})$, these values of m are still sparse. We conclude that our method is more useful than SOTA for large m.

2. *Special states*

To show our improvement for small m, we extract special states the DD representations of which are sparse and the

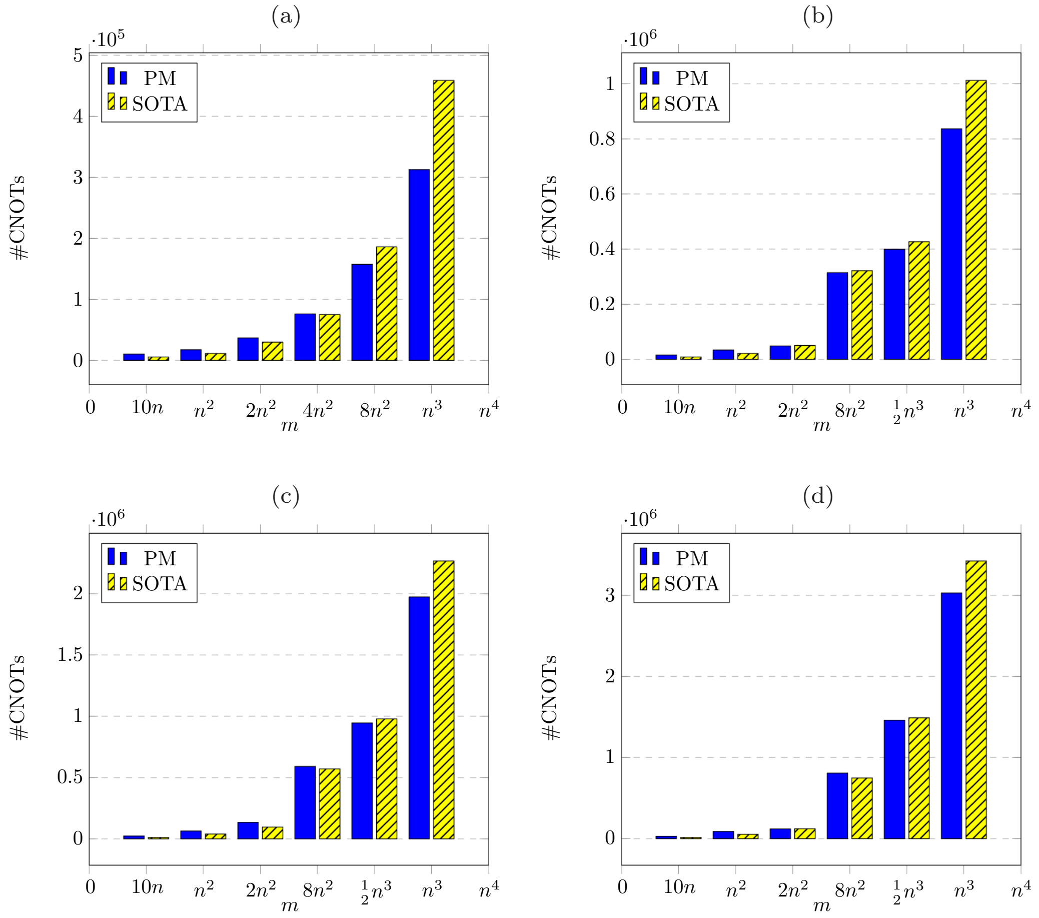

FIG. 5. Comparison between the CNOT complexities of PM and SOTA. PM is compared to the best-known algorithm (SOTA) in [28] on random sparse states of n qubits. For different n, we plot the number of CNOT gates required in both algorithms as a function of m, the number of nonzero amplitudes. It can be seen that PM requires fewer CNOTs in the interval between $2n^2$ and n^3 for $n = 16$ and 20, and between $8n^2$ and n^3 for $n = 25$ and 28. Moreover, increasing of m results in reduction of CNOTs. (a) $n = 16$. (b) $n = 20$. (c) $n = 25$. (d) $n = 28$.

reduction rules work well on them. These states are mostly uniform states that share paths better. These states benefit from the effect of reduced nodes which reduce the number of paths and branching nodes in each path. This results in reducing the number of multiple-controlled gates and their control qubits which is required for computing the ancilla qubit. Table I shows the average results for such states (set 1, 2, 3, and 4) in comparison with SOTA. We consider two different numbers of qubits 20 and 30, and small $m = n$ and $10n$. For each quantum state set, using the proposed method, we extract results regarding the number of nodes, number of reduced nodes, number of paths, and number of CNOTs. The number of reduced nodes shows that we can prepare several basis states together which reduces the number of CNOTs. Moreover, the number of paths, which is important in our complexity, is much less than the number of basis states, which results in reducing CNOTs. We also extracted the number of CNOTs by SOTA. Comparison shows that we reduce the number of CNOTs up to 98%.

QBA represents the quantum version of Byzantine agreement which works in constant time. In this protocol, for n players, we need to prepare the quantum state

$$|\varphi\rangle = \frac{1}{\sqrt{n^3}} \sum_{i=1}^{n^3} |i\rangle \tag{7}$$

on n qubits. For large n, this state is sparse. Table I shows its results. The proposed method prepares this state more efficiently. As shown in Table I, we reduce the number of CNOTs by 99.97% for QBA when $n = 30$. The reason is that the number of paths is much less than the number of nonzero basis states.

D. Algorithm performance

1. Correctness

First we explain how our algorithm prepares an arbitrary n-qubit state, given by Eq. (2), without any approximation error. It is enough to show that, starting from the initial state $|\text{no}\rangle_{q_A} \otimes |0\rangle^{\otimes n}$, in each iteration, in which the path $p_i \in P$ is traversed, we create a part $|\text{yes}\rangle \otimes \sum_{s\in S_{p_i}} \alpha_s|s\rangle$ of the target state, where S_{p_i} is the collection of basis states that are merged into path p_i in the creation of a DD.

Meanwhile, we keep the prepared parts $|\text{yes}\rangle \otimes \sum_{j<i} \sum_{s\in S_{p_j}} \alpha_s|s\rangle$ untouched. (Remember that $p_1 \succ p_2 \succ \cdots \succ p_k$.) In this way, after traversing the last path p_k, we end up with $|\text{yes}\rangle \otimes \sum_{j=1}^{k} \sum_{s\in S_{p_j}} \alpha_s|s\rangle$ as desired, where the system is in the target state and is uncorrelated with the ancillary qubit q_A.

To see how this is achieved in each iteration, first, notice that a path is uniquely characterized by its branching nodes and their values. For example, the path 000101 can be specified by $q_1 = 0$, $q_4 = 1$, $q_5 = 0$, and $q_6 = 1$, as in between q_1 and q_4 we adopt the convention that both q_2 and q_3 take the same value as q_1. Therefore, it is enough to prepare a branch without altering other branches, by acting on each node using its preceding branching nodes as the control. In our algorithm (more precisely, in the preparation rule), we further reduce the cost by the following crucial observation: When working on a qubit q in p_i, consider its closest ancestor the value of which is 1 in p_i, denoted by $\tilde{q}$. Since the sequence $p_1, p_2, \ldots, p_k$ is also ordered, only those completed parts (i.e., the partial state $\sum_{j<i} \sum_{s\in S_{p_j}} \alpha_s|s\rangle$) corresponding to paths $p_1, \ldots, p_{i-1}$ can have $\tilde{q} = |1\rangle$. On the other hand, for those paths where $\tilde{q} = |1\rangle$, they have already been completed and thus are tagged $|\text{yes}\rangle$ (regarded as $|0\rangle$ when used as a control qubit) on q_A. Therefore, it is sufficient to use two qubits ($\tilde{q}$ and q_A) as the control to make sure that other completed parts are unaltered in the course of preparing the ith part. As a result, we can complete the ith part without affecting the prepared paths by following the preparation rule of the algorithm. Since the branching nodes uniquely determine a path, we can flip the value of q_A of the ith part from $|\text{no}\rangle$ to $|\text{yes}\rangle$ by following the computing the ancilla rule.

2. Circuit complexity

In a DD, p_i and p_{i-1} may share a common subpath; therefore, we do not need to start preparation from the root for every p_i. This helps us to append fewer gates and reduces the number of CNOTs and single-qubit gates.

Our idea is based on a DD which we use as a reduced ordered BDD or ADD to represent the quantum state. Using them allows us to have a compact representation for the state and to remove redundancies that reduce circuit cost in the preparation. Moreover, reduced nodes help us to prepare some basis states together. Hence, in contrast to the previous works in which the number of basis states (m) is considered in the circuit complexity, the number of paths (k) is important in our complexity, and always

$$k \leqslant m. \tag{8}$$

According to Sec. II B, preparing a path is divided into two parts: preparing the path and computing the ancilla qubit. As a quantum circuit, it requires a sequence of 2-controlled gates to prepare the corresponding basis state (or basis states), and a multiple-controlled NOT gate to compute the ancilla qubit.

To compute the circuit complexity, we need to compute the number of 2-controlled gates for the first part, and the number of controls for the second part. The number of 2-controlled gates depends on the number of branching nodes in the path, and the number of one-children in the path of the corresponding basis state. Moreover, in the DD, paths overlap and we prepare each basis state from the last common node with the previous basis state instead of starting from the root. Considering this optimization, our algorithm reduces the number of 2-controlled gates. But in the worst case we require n 2-controlled gates. Decomposition of each 2-controlled gates requires four or six CNOTs, and so we need $O(n)$ CNOTs. For the second part, the number of controls is equal to the number of branching nodes in the path. Then, we make use of the method proposed in [30] to decompose the multiple-controlled NOT gate using $O(n)$ CNOT gates and one ancilla. We repeat the same procedure for k paths and so, in total, the number of CNOTs is equal to

$$\text{No. of CNOTs} = k \times O(n). \tag{9}$$

3. Time complexity

We traverse the DD twice to first compute G gates and secondly prepare the quantum state. As we visit each node once, each traversal is linear in the number of nodes, and such a number increases mildly (but not always) with problem size (i.e., qubits). The number of nodes depends on the number of paths and the number of qubits in each path. Hence, the number of nodes is always less than kn as there exist sharing nodes at least for the root. As a result, the classical runtime is less than $2kn$, which is less than the time required by the state of the art [28].

III. DISCUSSION

In this paper, we have proposed an algorithm to prepare quantum states deterministically. Our idea is based on preparing basis states one by one instead of operating one by one on the qubits. The latter is the key idea in general quantum state preparation algorithms. We have utilized DDs to represent quantum states in an efficient way. This allows our algorithm to be dependent on the number of paths where related works [26–28] are dependent on the number of basis states. We prepare the paths from the largest to the smallest regarding their binary bit strings. To do so, we traverse the DD in the preorder traversal. Through this traversal, we visit nodes on a path. For each node, depending on the existence of its two children (i.e., the *branching node*), we decide to either append 2-controlled single-target gates with different targets or just skip that. Upon preparing the path, an ancilla qubit is computed by adding a multiple-controlled NOT gate with the number of controls equal to the number of branching nodes in the path. Considering the decomposition method in [30], preparing each path and computing the ancilla qubit require $O(n)$ CNOTs. As a result, the final circuit cost depends on the number of paths and equals $O(kn)$. Using DDs helps us to have a compact representation of the state vector by reducing redundancies. The main advantages of our DD-based approach are as follows.

(1) For preparing each path, we do not need to start from the root node. We go back to the last common node with the previous path.

(2) When there are redundant nodes, removing them causes merging basis states to the same path and we can prepare them together. This helps in two ways. First, it reduces the number of iterations (k). Second, it reduces the number of branching nodes in paths which decreases the number of control qubits for computing the ancilla qubit.

Experimental results show that our idea works well for sparse DDs in which the numbers of paths and branching nodes are reduced. A sparse DD will be achieved when either m is small or m is not small but basis states share paths and can be prepared together. Hence, our algorithm besides SOTA works very well to prepare sparse states and states with sparse DDs. As future work, we can consider variable reordering in DDs to get a more sparse DD.

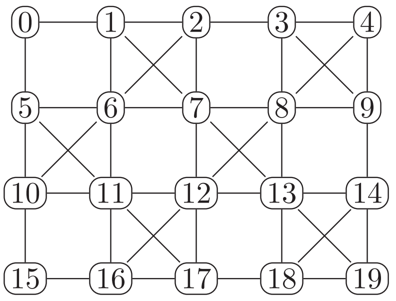

FIG. 6. Coupling map for the IBM Q Tokyo. Here 0, 1, . . . , 19 stand for physical qubits, and the edges indicate their connectivity.

As a concluding remark, we note that analyses in this paper are done assuming full connectivity between qubits, whereas a realistic *quantum processing unit* (QPU) is often subject to limited qubit connectivity. In the following, we compare our algorithm to SOTA with an example that takes into account the limited qubit connectivity.

Example 3. Consider preparing a uniform-amplitude quantum state corresponding to

$$S = \{1000,\ 0100,\ 0011,\ 0010,\ 0001,\ 0000\}. \quad (10)$$

To prepare it on a QPU with full qubit connectivity, our method and SOTA require 10 and 12 CNOTs, respectively. When preparing it on IBM's 20-qubit Tokyo with a coupling map as shown in Fig. 6, the cost depends on the mapping from logical qubits to physical qubits, which we choose to be

$$\{q_1 \to 0,\ q_2 \to 1,\ q_3 \to 6,\ q_4 \to 7\},\ q_A \to 2. \quad (11)$$

Under this mapping, compiling the circuit generated by our method and compiling the one generated by SOTA both result in two extra SWAP gates. As each SWAP is decomposed into three CNOTs, the final numbers of CNOTs for our method and SOTA are 16 and 18, respectively. Hence, for this example, our method outperforms SOTA both before and after the compilation.

The algorithm that we discussed in this paper is part of the ANGEL library [29], in the path "include/angel/quantum_state_preparation/." ANGEL is a c + + open-source library for quantum state preparation.

ACKNOWLEDGMENTS

This research was supported by the Google PhD Fellowship, by the Guangdong Basic and Applied Basic Research Foundation (Project No. 2022A1515010340), and by the University of Hong Kong Seed Fund for Basic Research for New Staff via Project No. 202107185045.

[1] S. Bravyi, D. Gosset, and R. König, Quantum advantage with shallow circuits, Science **362**, 308 (2018).

[2] C. Hempel, C. Maier, J. Romero, J. McClean, T. Monz, H. Shen, P. Jurcevic, B. P. Lanyon, P. Love, R. Babbush *et al.*, Quantum Chemistry Calculations on a Trapped-Ion Quantum Simulator, Phys. Rev. X **8**, 031022 (2018).

[3] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Quantum machine learning, Nature (London) **549**, 195 (2017).

[4] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, Simulated quantum computation of molecular energies, Science **309**, 1704 (2005).

[5] M. Plesch and Č. Brukner, Quantum state preparation with universal gate decompositions, Phys. Rev. A **83**, 032302 (2011).

[6] M. Mottonen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, Transformation of quantum states using uniformly controlled rotations, Quant. Inf. Comp. **5**, 467 (2005).

[7] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, Quantum circuits for isometries, Phys. Rev. A **93**, 032318 (2016).

[8] V. V. Shende, S. S. Bullock, and I. L. Markov, Synthesis of quantum-logic circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **25**, 1000 (2006).

[9] P. Kaye and M. Mosca, Quantum networks for generating arbitrary quantum states, in *International Conference on Quantum Information* (Optical Society of America, Rochester, New York, 2001), p. PB28.

[10] P. Niemann, R. Datta, and R. Wille, Logic synthesis for quantum state generation, in *46th International Symposium on Multiple-Valued Logic (ISMVL)* (IEEE, New York, 2016), pp. 247–252.

[11] V. V. Shende, I. L. Markov, and S. S. Bullock, Minimal universal two-qubit controlled-not-based circuits, Phys. Rev. A **69**, 062321 (2004).

[12] A. N. Soklakov and R. Schack, Efficient state preparation for a register of quantum bits, Phys. Rev. A **73**, 012307 (2006).

[13] Y. R. Sanders, G. H. Low, A. Scherer, and D. W. Berry, Black-Box Quantum State Preparation without Arithmetic, Phys. Rev. Lett. **122**, 020502 (2019).

[14] C. Zoufal, A. Lucchi, and S. Woerner, Quantum generative adversarial networks for learning and loading random distributions, npj Quantum Inf. **5**, 103 (2019).

[15] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, A divide-and-conquer algorithm for quantum state preparation, Sci. Rep. **11**, 1 (2021).

[16] R. Babbush, C. Gidney, D. W. Berry, N. Wiebe, J. McClean, A. Paler, A. Fowler, and H. Neven, Encoding Electronic Spectra in Quantum Circuits with Linear T Complexity, Phys. Rev. X **8**, 041015 (2018).

[17] X.-M. Zhang, T. Li, and X. Yuan, Quantum state preparation with optimal circuit depth: Implementations and applications, arXiv:2201.11495 (2022).

[18] F. Mozafari, M. Soeken, H. Riener, and G. De Micheli, Automatic uniform quantum state preparation using decision diagrams, in *50th International Symposium on Multiple-Valued Logic (ISMVL)* (IEEE, New York, 2020), pp. 170–175.

[19] F. Mozafari, H. Riener, M. Soeken, and G. De Micheli, Efficient boolean methods for preparing uniform quantum states, IEEE Transactions on Quantum Engineering **2**, 1 (2021).

[20] A. Bärtschi and S. Eidenbenz, Deterministic preparation of Dicke states, in *International Symposium on Fundamentals of Computation Theory* (Springer, New York, 2019), pp. 126–139.

[21] F. Mozafari, Y. Yang, and G. De Micheli, Efficient preparation of cyclic quantum states, in *27th Asia and South Pacific Design Automation Conference (ASP-DAC)* (IEEE, New York, 2022), pp. 460–465.

[22] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, Algebric decision diagrams and their applications, Formal methods in system design **10**, 171 (1997).

[23] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum Algorithm for Linear Systems of Equations, Phys. Rev. Lett. **103**, 150502 (2009).

[24] M. Ben-Or and A. Hassidim, Fast quantum byzantine agreement, in *Proceedings of the 37th Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, 2005), pp. 481–485.

[25] I. Streinu and L. Theran, Sparse hypergraphs and pebble game algorithms, European Journal of Combinatorics **30**, 1944 (2009).

[26] T. M. L. de Veras, L. D. da Silva, and A. J. da Silva, Double sparse quantum state preparation, Quantum Inf. Process **21**, 204 (2022).

[27] E. Malvetti, R. Iten, and R. Colbeck, Quantum circuits for sparse isometries, Quantum **5**, 412 (2021).

[28] N. Gleinig and T. Hoefler, An efficient algorithm for sparse quantum state preparation, in *Proceedings of the 58th ACM/IEEE Design Automation Conference* (IEEE, New York, 2021), pp. 433–438.

[29] ANGEL, https://github.com/fmozafari/angel.

[30] C. Gidney, Constructing large controlled NOTs, https://algassert.com/circuits/2015/06/05/Constructing-Large-Controlled-Nots.html (2015).