IEEE Transactions on Quantum Engineering

Received March 3, 2021; revised June 29, 2021; accepted July 20, 2021; date of publication August 4, 2021; date of current version August 30, 2021.

Digital Object Identifier 10.1109/TQE.2021.3101663

Efficient Boolean Methods for Preparing Uniform Quantum States

FERESHTE MOZAFARI (Member, IEEE), HEINZ RIENER (Member, IEEE), MATHIAS SOEKEN (Member, IEEE), AND GIOVANNI DE MICHELI (Life Fellow, IEEE)

Integrated Systems Laboratory, Ecole Polytechnique Federale de Lausanne, 1015 Lausanne, Switzerland

Corresponding author: Fereshte Mozafari (email: fereshte.mozafari@epfl.ch).

This work was supported in part by the Swiss National Science Foundation under Grant 200 021-169 084 MAJesty and in part by the Google Fellowship.

ABSTRACT As each quantum algorithm requires a specific initial quantum state, quantum state preparation is an important task in quantum computing. The preparation of quantum states is performed by a quantum circuit consisting of controlled-NOT (CNOT) and single-qubit gates. Known algorithms to prepare arbitrary n-qubit quantum states create quantum circuits in $O(2^n)$ runtime and use $O(2^n)$ CNOTs, which are more expensive than single-qubit gates in NISQ architectures. To reduce runtime and the number of CNOTs, we simplify the problem by considering an important family of quantum states, which are *uniform quantum states* (UQSs). We map UQSs to Boolean functions and propose a *UQS preparation* (UQSP) method. Preparing UQSs using Boolean functions allows us to utilize different representations of Boolean functions. We utilize decision diagrams to reduce runtime and enable preparation for a larger number of qubits where the state-of-the-art methods are not applicable. To further reduce the number of CNOTs, we utilize variable reordering and functional dependencies among the variables. Our state preparation method requires an exponential number of CNOTs in the worst case but it reduces CNOTs significantly for practical benchmarks. Moreover, our method generates an exact representation of quantum states without using free-qubits. We compare our algorithm with QisKit. The comparison shows that our UQSP method is capable to reduce the average number of CNOTs by 75.31% for the practical benchmarks. The runtime is almost reduced by a factor of 2.

INDEX TERMS Quantum compilation, quantum computing, quantum state preparation (QSP), uniform quantum states (UQSs).

I. INTRODUCTION

Many quantum algorithms assume a certain initial quantum state that needs to be loaded before the algorithm can perform its computation. A quantum state over n qubits is any superposition of all basis states $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$ represented by a vector of 2^n amplitudes. Each squared amplitude indicates the probability of the qubits being in the corresponding basis state. To load an initial quantum state, a quantum circuit has to be constructed that brings the qubits into the desired state. This process is called *quantum state preparation* (QSP).

Multiple algorithms [1]–[8] have been proposed for preparing *arbitrary quantum states*, which require an exponential number of CNOTs and runtime with respect to the number of qubits [9]. To alleviate this complexity, researchers either use free-qubits or prepare quantum states approximately—both ways add overheads. To remove these overheads, another way is to restrict the input space instead of considering arbitrary quantum states.

We restrict our search to a special family of quantum states, aiming for an efficient and exact implementation without using free-qubits. As an important family of quantum states, we consider *uniform quantum states* (UQSs) in this article. These states are superpositions of basis states, where all amplitudes are either zero or have the same value.

UQSs form an important class of quantum states and have recently attracted attention from researchers. For example, they have been used in effective quantum machine learning as the quantum version of a uniform random sample [10]. Many well-known quantum states are uniform, such as the uniform superposition of all basis states, the Bell state, the GHZ state [11], and the W state [12]. They also appear frequently

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

as initial states of important quantum algorithms, such as the quantum random-search algorithm *Grover walk* [13], *quantum Byzantine agreement* [14], and *secret sharing* [15], which have large applications in quantum cryptography. Hence, having an efficient algorithm for preparing UQSs alongside arbitrary quantum states is important.

In this article, we propose a new algorithm for *UQS preparation* (UQSP). The central idea of our work is a characterization of UQSs with Boolean functions. This characterization simplifies the problem because one simple Boolean function can describe an exponential number of amplitudes. Moreover, this enables us to apply well-understood and optimized techniques from logic synthesis [16]. We develop the theory and propose a functional decomposition method based on cofactoring for synthesis.

Our decomposition method can be applied to any representation of Boolean functions. We use truth-table-based, and decision-diagram-based representations for smaller, and larger numbers of qubits, respectively. In [17], we showed how we use decision diagrams to prepare UQSs. Decision diagrams enable a scalable quantum state preparation since many Boolean functions of practical interest have small representations, e.g., in terms of *binary decision diagrams* (BDDs) [18].

Moreover, whereas [17] targets runtime reduction using BDDs as a symbolic representation of Boolean functions, we present here an orthogonal improvement that identifies dependencies among variables of the Boolean functions and determines an order in which the qubits should be prepared. We show that, if a functional dependency can be identified, the decomposition algorithm can be avoided in favor of more efficient quantum circuit constructions. This idea allows us to reduce the overall number of CNOT gates.

We develop the underlying theory of UQSP using Boolean methods and demonstrate empirically that an implementation of our approach outperforms a state-of-the-art algorithm for preparing arbitrary quantum states [5] that is implemented in an industrial framework, QisKit [19]. Our algorithm achieves better quality (smaller circuits) in significantly less runtime.

The contributions of this article can be summarized as follows.

1) We propose a Boolean algorithm to prepare UQSs which simplifies the problem and enables us to apply well-understood techniques from logic synthesis.
2) We apply our decomposition algorithm to the BDD representation of Boolean functions to enable a fast execution when the function representation is small (the algorithm runs in polynomial time with respect to the number of BDD nodes). Experimental results show that by increasing the number of qubits to at most 30, our method prepares states in milliseconds whereas QisKit cannot prepare states for more than 18 qubits.
3) We investigate the problems of identifying functional dependencies among the qubits and determining the best order for preparing the qubits of a UQS. We present several heuristics for solving these two problems and show that we can construct quantum circuits for UQSP with different tradeoffs between runtime and the number of CNOTs. The comparison with QisKit shows that the proposed method achieves an average reduction on the number of CNOTs by 75.31% for practical benchmarks and the runtime is improved by almost a factor of 2. Hence, our algorithm for preparing UQSs can be integrated into QisKit to improve the quality-of-results for this important family of quantum states.

This article is organized as follows. In Section II, we present related works. In Section III, we explain some preliminaries on Boolean functions and quantum circuits. Our motivation for preparing UQSs is shown in Section IV. In Section V, we propose our Boolean algorithm using functional decomposition. Next, in Sections VI and VII, we present our optimization methods using decision diagrams and functional dependency analysis, respectively. We evaluate our methods in Section VIII and compare their results against QisKit. Finally, Section IX concludes this article.

II. RELATED WORKS

The general problem of preparing arbitrary quantum states requires quantum circuits with an exponential number of CNOTs, and exponential depth and runtime in the worst case. Multiple algorithms have been proposed for preparing them.

The algorithm presented in [8] prepares quantum states with a divide-and-conquer strategy. Although it creates quantum circuits with polylogarithmic depth, it uses additional n free-qubits and increases the number of elementary quantum gates. In [20], the authors present a way of preparing arbitrary quantum states with l unique amplitudes for the purpose of reducing T gates. To prepare a state, they start with a uniform superposition. If l is not a binary power, they prepare the initial superposition using amplitude amplification. Afterward, by loading data from *quantum read-only memory* (QROM), they decide whether to keep the state or to alter it. Moreover, they prepare states with the help of free-qubits. The authors in [21] propose an algorithm to prepare only efficiently integrable probability distributions. Their algorithm requires overheads by using additional free-qubits.

The authors in [6] and [7] propose algorithms without using free-qubits, but they prepare quantum states approximately. Approximate state preparation algorithms introduce some inaccuracy, such that all computations are only correct as long as a certain error threshold is not exceeded.

In this article, we are interested in algorithms that prepare the initial quantum state exactly and without using free-qubits, which reduce the implementation overhead. These algorithms [1]–[5] to prepare arbitrary quantum states, however, require an exponential number of CNOTs and runtime with respect to the number of qubits [9]. To reduce runtime and the number of CNOTs, we propose a Boolean algorithm and compare our results with [5] from this category.

III. PRELIMINARIES

A. BOOLEAN FUNCTIONS

Let $\mathbb{B} = \{0, 1\}$. A *Boolean function* $f : \mathbb{B}^n \to \mathbb{B}$ of n Boolean variables $x = x_0, \ldots, x_{n-1}$ is a mapping of n Boolean input values $\hat{x} = \hat{x}_0, \ldots, \hat{x}_{n-1}$ to a single Boolean output value $f(\hat{x})$. Each Boolean function is given as input 2^n input assignments.

A *literal* is a Boolean variable $x_i^1 = x_i$ or its negation $x_i^0 = \bar{x}_i$. A *product term* (or *cube*) $t(x)$ is a conjunction of literals. The size $|t|$ of a product term t is the number of literals appearing in t, and a product term in which all Boolean variables $x_0, \ldots, x_{n-1}$ appear exactly once is a *minterm*. Each Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ can be uniquely represented in its *minterm canonical form*

$$\begin{aligned} f(x) = \quad & f(0, \ldots, 0) \cdot \bar{x}_{n-1} \cdots \bar{x}_0 + \\ & \vdots \\ & + f(1, \ldots, 1) \cdot x_{n-1} \cdots x_0 \end{aligned} \tag{1}$$

where $f(\hat{x}_0, \ldots, \hat{x}_{n-1})$ are Boolean values, called *discriminants*, and $x_{n-1}^{\hat{x}_{n-1}} \ldots x_0^{\hat{x}_0}$ are product terms. Consequently, the set

$$\mathsf{Minterms}(f) = \{x_{n-1}^{\hat{x}_{n-1}}, \ldots, x_0^{\hat{x}_0} | f(\hat{x}_0, \ldots, \hat{x}_{n-1}) = 1\} \tag{2}$$

of all minterms of a Boolean function f characterizes the function uniquely. A Boolean function f is represented by its minterms. The size $|f|$ of f is defined as the number of minterms of f.

The *positive cofactor* f_{x_i} and *negative cofactor* $f_{\bar{x}_i}$ of a Boolean function f denote f restricted to the subdomain in which x_i takes the value 1 and 0, respectively. We use cofactors to compute the influence of each variable on the function's output. To compute the probability of x_i being 1 and 0 in f, we define

$$p_f(x_i) = \frac{|f_{x_i}|}{|f|} \quad \text{and} \quad p_f(\bar{x}_i) = \frac{|f_{\bar{x}_i}|}{|f|} \tag{3}$$

respectively. Note that $p_f(x_i) + p_f(\bar{x}_i) = 1$. Moreover, we utilize *Shannon's decomposition* theorem, which states $f = x_i f_{x_i} + \bar{x}_i f_{\bar{x}_i}$ for variables x_i of f. The intuition is that Shannon's decomposition uses cofactors to partition a function into two halves.

Each Boolean function can also be represented in many semantically equivalent but structurally different *exclusive-or sum-of-products* (ESOP) forms

$$f(x) = t_0(x) \oplus \cdots \oplus t_{m-1}(x) \tag{4}$$

where $t_0, \ldots, t_{m-1}$ are product terms and m is a positive integer called the *degree* of the ESOP form.

ESOP synthesis is the problem of finding an ESOP form for a Boolean function f subject to a cost function. Exact algorithms, such as [22] and [23], and heuristics, such as [24], have been proposed for finding and minimizing the degree of an ESOP form of a Boolean function f. These algorithms have applications in quantum compilation flows since they allow us to decompose multiple-controlled single-target gates into a sequence of generalized Toffoli gates [25], [26].

There are different representations for Boolean functions but we consider representations as truth table and *binary decision diagram* (BDD) in this article. For both of them, we can compute Shannon's decomposition efficiently. The truth table of a Boolean function is a tabulation of its value at each of the 2^n input assignments. The BDD of a Boolean function is a symbolic representation that shows a compact representation for it.

A BDD [27] is a directed acyclic graph $(V \cup f \cup \{0, 1\})$ representing a Boolean function f, where V is the set of nodes. For each $v \in V$, the outdegree is two. For the root node, the indegree is one. f points to the root node of the BDD. $\{0, 1\}$ are terminal nodes with zero outdegrees. The two outgoing arcs of a node $v \in V$ are labeled as T and E. $T(v)$ and $E(v)$ correspond to the one-child and zero-child and are represented by a solid line and a dashed line, respectively.

B. QUANTUM CIRCUITS

A *quantum bit* (qubit) is the elementary unit of information in quantum computation. A *quantum state* $|\varphi\rangle$ over n qubits is characterized by

$$|\varphi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{5}$$

a column vector of 2^n *amplitudes* $\alpha_i \in \mathbb{C}$ with $|\alpha_0|^2 + \cdots + |\alpha_{2^n-1}|^2 = 1$. Each squared amplitude $|\alpha_i|^2$ indicates the probability that after measurement the n qubits are in the classical state i.

A *quantum circuit* is a structural description of a quantum program in terms of quantum gates connected by quantum wires, indicating the passing of time. A *quantum gate* is an operation applied to one or more qubits to change their quantum state. A quantum gate that acts on n qubits can be specified by a $2^n \times 2^n$ unitary matrix U [28], [29]. A matrix U is *unitary* if

$$U^\dagger U = U U^\dagger = I \tag{6}$$

where $U^\dagger$ is the transposed complex conjugate of U. The *matrix product* $A \cdot B$, the *direct sum* $A \oplus B$, and the *direct product* (also called *Kronecker product*) $A \otimes B$ of two matrices A and B, respectively, are defined as usual [29].

In this article, we divide elementary quantum gates into two classes: The *single-qubit gates* with unitary matrices

$$R_x(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\mathrm{i}\sin\frac{\theta}{2} \\ -\mathrm{i}\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}, \quad R_y(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$R_z(\theta) = \begin{pmatrix} e^{-\mathrm{i}\frac{\theta}{2}} & 0 \\ 0 & e^{\mathrm{i}\frac{\theta}{2}} \end{pmatrix} \tag{7}$$

with parameter $\theta \in [0, 2\pi]$

$$\text{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \tag{8}$$

and the *controlled-NOT gate* (CNOT) with unitary matrix

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{9}$$

Additionally, we use the family of *uniformly controlled single-target gates* [1], [30] that correspond to $2^n \times 2^n$ matrices

$$U_i = u_0 \oplus \cdots \oplus u_{k-1} = \begin{pmatrix} u_0 & & 0 \\ & \ddots & \\ 0 & & u_{k-1} \end{pmatrix} \tag{10}$$

with one target qubit q_i and $n-1$ control lines. The matrices can be decomposed into the direct sum of $k = 2^{n-1}$ unitary 2×2 matrices $u_0, \ldots, u_{k-1}$.

IV. UQSP MOTIVATION

A. UQSP PROBLEM

In this work, we consider n-qubit quantum states that are uniform superpositions over a nonempty subset of the basis states $|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle$. In such quantum states all amplitudes of the state vector are either 0 or have the same value $\alpha = 1/\sqrt{s}$, where s is the size of the subset of basis states. We exploit the fact that such states can be characterized by a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$, such that $f(x) = 1$, if and only if $|x\rangle$ is in the subset of the considered basis states, and therefore its corresponding amplitude is nonzero.

Example 1: The uniform quantum state

$$|\varphi\rangle = \frac{1}{\sqrt{3}}(0, 1, 1, 0, 1, 0, 0, 0)^T \tag{11}$$

which is the W state over 3 qubits, corresponds to the Boolean function f_W that has the truth table

$$f_W(x_0, x_1, x_2) = (0, 1, 1, 0, 1, 0, 0, 0)^T. \tag{12}$$

Proposition 1: There is a one-to-one correspondence between uniform quantum states and Boolean functions. A uniform quantum state $|\varphi_f\rangle$ corresponds to the normalized truth table of f

$$|\varphi_f\rangle = \frac{f}{\sqrt{|f|}} = \frac{1}{\sqrt{|f|}} \sum_{x \in \text{Minterms}(f)} |x\rangle.$$

Now, the UQSP problem refers to the problem of finding a quantum circuit that prepares such a state, given as input a Boolean function f in some representation. In the beginning, all qubits are assumed to be in their zero states. Then, we are searching for a construction that transforms a given unitary matrix UQSP(f) into a circuit, where

$$\text{UQSP}(f)|0\rangle^{\otimes n} = |\varphi_f\rangle. \tag{13}$$

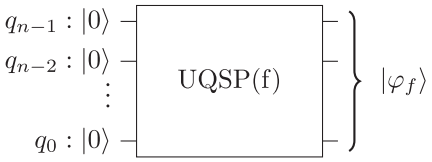


FIGURE 1. Problem of preparing the UQS corresponding to the given Boolean function f as input.

Fig. 1 describes our problem formulation.

B. MOTIVATIONAL EXAMPLES

UQSs are used in several protocols in quantum communication and cryptography, for example, in the *Quantum Byzantine Agreement* (QBA) [14], [31]. Byzantine agreement protocols are important algorithms that are robust to failures in distributed computing. A group of n players must agree on a bit despite the faulty behavior of some of the players. QBA represents the quantum version of Byzantine agreement which works in constant time [14]. It is shown that in this protocol, for n players, we are required to prepare two quantum states that are uniform, namely

$$|\varphi_1\rangle = \frac{1}{\sqrt{2}}(|0, 0, \ldots, 0\rangle + |1, 1, \ldots, 1\rangle) \tag{14}$$

on n qubits, and

$$|\varphi_2\rangle = \frac{1}{\sqrt{n^3}} \sum_{a=1}^{n^3} |a\rangle \tag{15}$$

on n qubits.

Although the quantum state in (14) represents the well-known GHZ state and we can apply a template to prepare it, the second state in (15) is more general and we require an automated algorithm to prepare it.

As another example, the initial quantum state used in the quantum coupon collector (QCC) [10] problem is also uniform. It is given as

$$|\varphi\rangle = \frac{1}{\sqrt{|S|}} \sum_{i \in S} |i\rangle \tag{16}$$

over the elements of an unknown k-element set $S \subseteq \{1, \ldots, n\}$, where $k = |S| < n$.

Moreover, UQSs are helpful when we want to extend a problem. In this domain, we already have the results for some parts of the problem and we only need to solve the problem for a new part. Instead of preparing all possible input assignments, we only need to prepare the input assignments for the new part that are UQSs. As an example, consider the Zed city problem introduced in [32], which is an instance of vertex coloring. As some of the nodes are colored already, it requires a UQS in the beginning to create the desired input assignments for uncolored nodes.

Hence, all these applications show that having an automated algorithm to prepare UQSs efficiently is very important.

IEEE Transactions on Quantum Engineering

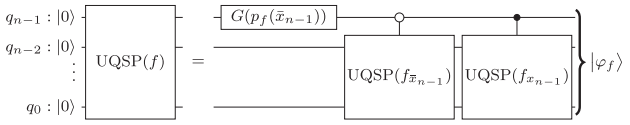


FIGURE 2. Preparation of Boolean function *f* using functional decomposition.

V. USING FUNCTIONAL DECOMPOSITION FOR UQSP

To prepare an n-qubit uniform quantum state $|\varphi_f\rangle$ that encodes the Boolean function f, we define a correspondence between the qubits in the quantum circuit and the variables in f. Hence, the uniform quantum state $|\varphi_f\rangle$ and the qubit q_i correspond to the Boolean function f and the variable x_i, respectively. In the remainder of this article, we will use these symbols interchangeably.

Representing uniform quantum states as Boolean functions allows us to apply Shannon's decomposition to solve the state preparation problem recursively.

To construct the desired quantum circuit corresponding to the UQSP(f) block shown in Fig. 1, we iterate over the variables of the Boolean function in an order, and prepare them one by one by applying the Shannon's decomposition. For each qubit q_i (x_i), we decompose UQSP blocks as follows.

1) A gate $G(P)$ is applied, which is a unitary transformation and satisfies
$$G(P)|0\rangle = \sqrt{P}|0\rangle + \sqrt{1-P}|1\rangle. \tag{17}$$
The parameter P represents the probability of $q_i(x_i)$ being zero in the current decomposed function ($\hat{f}$), which equals to $p_{\hat{f}}(\bar{x}_i)$.
2) q_i is either zero or one. Hence, we construct new blocks for the corresponding cofactors by applying negative-control and positive-control, respectively.

Fig. 2 shows the construction of UQSP(f) using our functional decomposition for one iteration, where $i = n-1$. We formulate the general idea of our state preparation algorithm as

$$\text{UQSP}(f)|0\rangle^{\otimes n} = (\text{UQSP}(f_{\bar{x}_i}) \oplus \text{UQSP}(f_{x_i}))(G(p_f(\bar{x}_i)) \otimes I_{2^n-1})|0\rangle^{\otimes n}. \tag{18}$$

By applying this procedure recursively for all variables, we obtain the desired quantum circuit.

Example 2: Applying the proposed functional decomposition algorithm for the Boolean function of Example 1 (f_W) in the order x_2, x_1, x_0, results in Fig. 3 . First, we decompose f over the variable x_2. We apply $G(p_f(\bar{x}_2))$ and divide the circuit into two parts corresponding to cofactors $f_{\bar{x}_2}$ and f_{x_2} by adding negative and positive controls, respectively. Next, we decompose with x_1, leading to four cofactors $f_{\bar{x}_2\bar{x}_1}$, $f_{\bar{x}_2x_1}$, $f_{x_2\bar{x}_1}$, and $f_{x_2x_1}$. Finally, the four cofactors are decomposed with x_0. The detailed structure is shown on the left-hand side of the figure. It is visible that preparing each qubit requires 2^k *MC*-gates with k controls with different polarities corresponding to different cofactors.

By applying the definition in (3), the probability values for G gates are computed as the right-hand side of the figure. The target-qubit represented by $G(-)$ specifies a division by zero corresponding to the zero-probability ($G(\frac{0}{0})$). This case never happens and we do not insert any gate. Moreover, $G(1)$ shows that the qubit is always equal to the zero state and can be safely ignored, too.

The resulted quantum circuit consists of a sequence of multiple-controlled single-target gates with $G(P)$ as target. From the definition of $R_y(\theta)$ and (17) one can readily derive that

$$G(P) = R_y\left(2\cos^{-1}(\sqrt{P})\right). \tag{19}$$

Consequently, by replacing all gates on the target-qubit by R_y gates, we obtain a circuit consisting only of *multiple-controlled* R_y (MC-R_y) gates.

Example 3: Fig. 4 shows quantum gate realization of G gates for UQSP(f_W) in Fig. 3 using (19).

VI. LOGIC SYNTHESIS TECHNIQUE 1: USING DECISION DIAGRAMS

In practice, it is infeasible to store Boolean functions using truth tables for a large number of variables (typically more than 15 variables). As an alternative, the proposed approach can extract the quantum circuit for UQSP(f) when f is represented as a *reduced ordered BDD* (ROBDD, or BDD for short) [27]. This is due to the fact that counting all minterms and computing cofactors can be efficiently performed using BDDs. The compact representation not only enables a scalable quantum state preparation, if the BDD representation for f is small, but also reduces the number of multiple-controlled single-target gates and maybe CNOTs.

Algorithm 1 shows the UQSP using decision diagrams. First, in procedure *ComputeZeroProbabilities*, we traverse the BDD in top-down post-order to compute zero probabilities by dividing the number of ones of the zero-child with the number of ones of the current node. The number of ones of the current node is equal to the summation of the number of ones of the zero-child and the one-child. To compute the number of ones of the zero-child and the one-child, we consider the effect of reduced nodes between these nodes and the current node using *ComputeOnesFromChild*. Second, in procedure *ComputeMCgates*, we traverse again in top-down post-order to extract multiple-controlled single-target gates (MC-gates). For each node, we insert a G gate with its zero probability. Next, we connect a negative-control and a positive-control to the gates from the zero-child and the one-child, respectively, using procedure *ApplyControlToChildGates*. We also consider the effect of the reduced nodes in the path between the current node and its children. Both children of the reduced nodes have ones in the Boolean function, so we insert gates with 1/2 probabilities in procedure *InsertHalfGatesForRNodes*. Finally, at the root-node, we have all MC-gates.

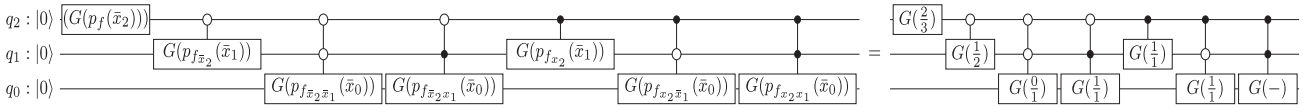


FIGURE 3. Multiple-controlled single-target gates of UQSP(f_W).

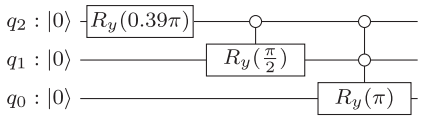


FIGURE 4. Sequence of MC-R_y gates for UQSP(f_W).

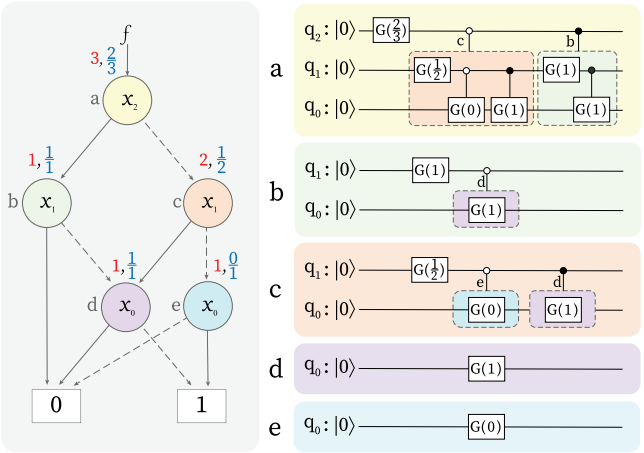


FIGURE 5. BDD representation of f_W and the procedure of the preparing it.

Example 4: We illustrate the BDD-based synthesis algorithm for the f_W function as an example in Fig. 5 . The BDD consists of 5 nodes. We traverse the BDD in top-down post-order to count the number of ones and the zero probabilities by dividing the number of ones from the zero-child (dashed line) by the number of ones of the current node, for each node. The number of ones and the probabilities for each node are shown within the figure on the left-hand side in red and blue colors, respectively.

To construct MC-gates, we again traverse the BDD in top-down post-order. In a recursive manner, we construct for each node a circuit that consists of one $G(P)$ gate, a negative-controlled application of the circuit constructed by the zero-child, as well as a positive-controlled application of the circuit constructed by the one-child. These gates are located in the figure using boxes at the right-hand side of each node with the same color. As an example, we explain the construction of the gates for node "*b*" (green color). First, we have to apply $G(\frac{1}{1})$ to the qubit corresponding to "*b*," then we connect the negative-control to the gates of node "*d*." The one-child is connected to constant zero, meaning that it is not included in the minterms of the function and we do not need to consider it, so we do not insert any gate for the one-child.

Algorithm 1: UQSP Using Decision Diagrams.

Input: The root r of the BDD of the Boolean function f
Output: Quantum circuit QC
Proc $UQSP_{DD}(r)$**:**
  $p = \{\}$
  $ones = \{\}$
  ComputeZeroProbabilities$(r, p, ones)$
  $QC = \{\}$
  ComputeMCgates(r, p, QC)
  return $QC[r]$

Proc *ComputeZeroProbabilities*$(v, p, ones)$**:**
  if *IsTerminal*(v) *or IsVisited*(v) **then**
    return
  ComputeZeroProbabilities$(E(v), p, ones)$
  ComputeZeroProbabilities$(T(v), p, ones)$
  E_ones = ComputeOnesFromChild$(v, E(v), ones)$
  T_ones = ComputeOnesFromChild$(v, T(v), ones)$
  $p[v] = \frac{E\_ones}{E\_ones+T\_ones}$
  $ones[v] = E\_ones + T\_ones$
  return

Proc *ComputeOnesFromChild*$(v, c, ones)$**:**
  if *IsZeroTerminal*(c) **then**
    return 0
  R_nodes = NumberOfReducedNodes(v, c)
  if *IsOneTerminal*(c) **then**
    return 2^{R_nodes}
  return $ones[c] \times 2^{R\_nodes}$

Proc *ComputeMCgates*(v, p, QC)**:**
  if *IsTerminal*(v) *or IsVisited*(v) **then**
    return
  ComputeMCgates$(E(v), p, QC)$
  ComputeMCgates$(T(v), p, QC)$
  InsertGgate$(p[v]$, QC$[v])$
  ApplyControlToChildGates$(v, E(v)$,'Negative', $QC)$
  ApplyControlToChildGates$(v, T(v)$,'Positive', $QC)$
  InsertHalfPGatesForRNodes$(v, E(v)$,'Negative', $QC)$
  InsertHalfPGatesForRNodes$(v, T(v)$,'Positive', $QC)$
  return

Proc *ApplyControlToChildGates*$(v, child, control\_type, QC)$**:**
  for $i = 0, \ldots, n-1$ **do**
    foreach MCgate with G *on* $q_i \in QC[child]$ **do**
      MCgate. AddControl(control_type, Index(v))
      QC$[v]$.AddMCgate(MCgate)
  return

Proc *InsertHalfPGatesForRNodes*$(v, child, control\_type, QC)$**:**
  for $i = Index(v) + 1, \ldots, Index(child)$ **do**
    MCgate = CreateMCgate$(\frac{1}{2}, q_i)$
    MCgate. AddControl(control_type, Index(v))
    QC$[v]$.AddMCgate(MCgate)
  return

VII. LOGIC SYNTHESIS TECHNIQUE 2: USING DEPENDENCY ANALYSIS

For each qubit q_i, the recursion generates 2^k MC-gates. The variable k denotes the number of already prepared qubits that

IEEE Transactions on Quantum Engineering

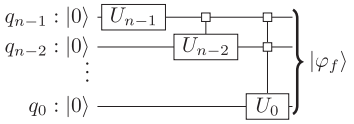


FIGURE 6. General structure of the sequential preparation of qubits using uniformly controlled single-qubit gates.

can be used as control qubits for preparing the current qubit q_i. The variable k can range from 0 to $n-1$.

This sequence of multiple-controlled single-target gates on the same target-qubit can be fused into a uniformly controlled single-target gate U_i per qubit. Applying this procedure for all variables results in the structure depicted in Fig. 6, where each qubit is prepared with a uniformly controlled single-target gate in the order from $n-1$ to 0.

On one hand, the structure of the quantum circuit presented in Fig. 6 shows that preparing qubits depends on all previously prepared qubits. On the other hand, formulating the UQSP problem using Boolean functions allows us to identify functional dependencies among variables. If a dependency function is recognized, meaning that a qubit depends on a subset of previously prepared qubits, then often a more compact quantum circuit structure can be synthesized. This helps us to further reduce the number of control qubits and CNOTs. An identified functional dependency for q_i can be utilized in three ways: 1) to reduce the number of control qubits if q_i depends only on a subset of the previously prepared qubits, 2) to reduce the number of elementary quantum gates if the functional dependency can be well expressed with a library of hardware-supported quantum gates, and 3) to reduce the number of control qubits for preparing other next qubits.

Algorithm 2 shows the UQSP using functional dependencies (UQSP_{FD}) in pseudocode. As inputs, it takes a Boolean function f that defines a uniform quantum state and two strategies R_A and D_A for variable reordering and dependency analysis. Both strategies can be either implemented as exact or heuristic algorithms, which allow us to choose between different runtime-quality tradeoffs.

From a bird's-eye perspective, UQSP_{FD}applies the dependency analysis algorithm D_A to the Boolean function f and all reordered functions f' of f suggested by R_A. The algorithm then synthesizes a quantum circuit by considering dependency functions extracted from D_A, counts the number of CNOTs, and returns the best quantum circuit realized for the function. The considered cost function focuses on reducing CNOTs which correlates with the reduction of rotation gates, such that the algorithm minimizes CNOTs and rotation gates.

The procedure *SynthesizeQC$_{FD}$* shows how a quantum circuit is realized from a Boolean function f with a fixed order of variables and a fixed set of functional dependencies D. The procedure iterates one by one over the variable indices, checks if a dependency function for this index is in D, and then either synthesizes the quantum circuit for the given dependency function or uses the recursive decomposition of

Algorithm 2: UQSP Using Functional Dependencies.

Input: Boolean function f, dependency analysis algorithm D_A, variable reordering algorithm R_A
Output: Quantum circuit QC, qubits order
Proc $UQSP_{FD}(f, R_A, D_A)$**:**
  QC = SynthesizeQC$_{\text{FD}}(f, D_A(f))$
  $cost$ = CNOTs(QC)
  $f_{\text{best}} = f$
  foreach reordered $f' \in R_A(f)$ **do**
    $QC_{f'}$ = SynthesizeQC$_{\text{FD}}(f', D_A(f'))$
    $cost_{f'}$ = CNOTs($QC_{f'}$)
    if $cost_{f'} < cost$ **then**
      $QC = QC_{f'}$
      $cost = cost_{f'}$
      $f_{\text{best}} = f'$
  return QC, Order(f_{best})

Proc `SynthesizeQC`$_{\text{FD}}$ (f, D)**:**
  QC = CreateNewQC()
  n = NumberOfVariables(f)
  for $i = n-1, \ldots, 0$ **do**
    QC.CreateQubit(i)
    d_i = D.FindDependency(i)
    if $d_i \neq \perp$ **then**
      QC.CreateGatesForDependencyFunction(i, d_i)
    else
      QC.CreateGatesRecursively(i)
  return QC

(18). Dependency functions are only computed if they are guaranteed to reduce the number of CNOTs when compared to the recursive decomposition.

The detailed explanation of dependency analysis methods (D_A), CNOTs cost functions, and variable reordering methods (R_A) are described next.

A. DEPENDENCY ANALYSIS METHODS

Suppose that $f : \mathbb{B}^n \to \mathbb{B}$ is a Boolean function over Boolean variables $x = x_0, \ldots, x_{n-1}$ and γ is a cost function that assigns integers to Boolean operators. We attempt to compute a series $g_0, \ldots, g_{n-1}$ of dependency functions $g_i(x_{i+1}, \ldots, x_{n-1})$, such that

$$x_i \cdot f(x) = g_i(x_{i+1}, \ldots, x_{n-1}) \cdot f(x) \text{ for } i = 0, \ldots, n-2 \tag{20}$$

to minimize

$$\gamma = \sum_{i=0}^{n-1} \gamma(g_i). \tag{21}$$

As the cost function, we consider the number of CNOTs required to prepare the qubits with the dependency functions or the recursive construction of (18) if no dependency function exists.

We propose two algorithmic strategies to compute dependency functions: 1) Pattern search and 2) SAT-based ESOP synthesis. For a given Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ over Boolean variables $x = x_0, \ldots, x_{n-1}$, both strategies iterate over the x_is for $0 \leq i \leq n-1$ and attempt to identify one dependency function g_i.

1) *Pattern Search*. We iterate over all k-tuples $(v_0, \ldots, v_{k-1})$ of the set $\{x_{i+1}, \ldots, x_{n-1}\}$ over Boolean variables for increasing values of k, $1 \leq k \leq K$, where K is a fixed upper bound, and test if

$$x_i \cdot f(x) = g(v_0, \ldots, v_{k-1}) \cdot f(x) \tag{22}$$

for all dependency functions g from some fixed set of patterns. If the test succeeds, $g(v_0, \ldots, v_{k-1})$ is a dependency function for x_i. We consider three different types of dependency functions: The identity function with a single argument to identify a dependency on a single variable or its negation, the k-ary XOR function to identify XOR relations between multiple variables or their negations, and the k-ary AND function to identify AND relations between multiple variable or their negations.
2) *SAT-Based ESOP Synthesis*. We attempt to compute a dependency function in two steps.
Determining functional support: In the first step, we decide on the support of the dependency function using distinguishing bit-pairs [33]. We compute the distinguishing bit-pairs of $t(x) = x_i \cdot f(x)$ and sort the variables $x_{i+1}, \ldots, x_{n-1}$ by the number of distinguished bit-pairs with respect to $t(x)$. We accumulate a set $S \subseteq \{x_{i+1}, \ldots, x_{n-1}\}$ of variables in this order until $t(x)$ is guaranteed to be "reconstructable" using S, i.e., for each distinguishing bit-pair in $t(x)$, there is a variable in S with the same distinguishing bit-pair.
Synthesizing structure: In the second step, we use SAT-based synthesis [23] to derive an ESOP form g of the dependency function with support S that satisfies (20). Note that our cost function prioritizes XORs with many fanins and ANDs with few fanins.

B. CNOT COSTS

We define a cost function γ that counts the number of CNOTs required to realize a dependency function $g_i(x)$ as a quantum circuit based on the general constructions proposed by Schuch and Siewert [34], Welch *et al.* [35], and Mottonen *et al.* [1]. We exploit the fact that, since in the beginning, all qubits are zero, the relative phase of the rotation gates is zero, which allows us to reduce the number of gates.

We distinguish three cases as follows.

1) *XOR Clause.* Let $g_i(x) = t(x)$ be an m-ary XOR clause $l_0 \oplus \cdots \oplus l_{m-1}$ of literals l_j, $0 \leq j \leq m-1$. The qubit q_i can be prepared by a quantum circuit using $\gamma_1(t)$ CNOTs, where

$$\gamma_1(t) = |t| \tag{23}$$

is the number of literals.
2) *Product Term.* Let $g_i(x) = t(x)$ be a product term $t(x) = l_0 \cdots l_{m-1}$ of literals l_j, $0 \leq j \leq m-1$. The qubit q_i can be prepared using a quantum circuit which requires $\gamma_2(t)$ CNOTs, where

$$\gamma_2(t) = \begin{cases} |t| & \text{if } |t| \in \{0, 1\} \\ 2^{|t|} & \text{if } |t| > 1. \end{cases} \tag{24}$$

In the case of no dependency, the number of CNOTs is zero. When the number of literals is one, it indicates that there is an equality dependency, so the number of CNOTs is one. For more than one literal, we express the dependency pattern with a multiple-controlled Toffoli gate. If we consider a multiple-controlled Toffoli gate and some identity gates, we can utilize the decomposition method presented in [1] for uniformly controlled single-target gates. Applying this decomposition method results in $2^{|t|}$ CNOTs.
3) *ESOP Form.* Let $g_i(x) = t_0(x) \oplus \cdots \oplus t_{m-1}(x)$ be an ESOP form of product terms $t_j(x)$, $0 \leq j \leq m-1$. We express the dependency pattern by a sequence of multiple-controlled Toffoli gates. Then, we propose two different ways to decompose this sequence, which require different numbers of CNOTs.
Multiple-controlled Toffoli gate decomposition prepares the qubit q_i using $\gamma_3(t_0 \oplus \cdots \oplus t_{m-1})$ CNOTs, where

$$\gamma_3(t_0 \oplus \cdots \oplus t_{m-1}) = \gamma_2(t_{j^\star}) - \gamma_4(t_{j^\star}) + \sum_{i=0}^{m-1} \gamma_4(t_i) \tag{25}$$

with

$$\gamma_4(t_i) = \begin{cases} |t_i| & \text{if } |t_i| \in \{0, 1\} \\ 2^{|t_i|+1} - 2 & \text{if } |t_i| > 1 \end{cases} \tag{26}$$

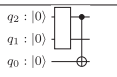
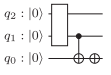
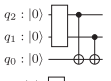
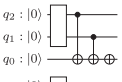
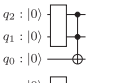
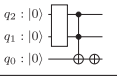
and

$$j^\star = \underset{i=0,\ldots,m-1}{\operatorname{argmax}} |\gamma_2(t_j) - \gamma_4(t_j)|. \tag{27}$$

As the initial state is zero in the beginning, we decompose the first gate using [1] with fewer CNOTs and its cost is represented in (24). For the rest we use the methods presented in [34] and [35] that do not require free-qubits. Their corresponding cost is represented in (26). To reduce the number of CNOTs, we bring the gate with more controls to the beginning and its index is computed by (27). As the summation $\sum_{i=0}^{m-1} \gamma_4(t_i)$ in (25) accumulates the cost for all gates, the first gate's cost ($\gamma_4(t_{j^\star})$) is subtracted from the result.
Uniformly controlled single-target gate decomposition is applicable if the literals of the product terms t_j are subsets of each other. Then, they can be considered as controls of a uniformly controlled single-target gate. In comparison to the recursive construction, using uniformly controlled single-target gates leads to fewer controls and reduces the number of CNOT gates, which are computed as follows:

$$\gamma_5(t_0 \oplus \cdots \oplus t_{m-1}) = \gamma_2(t_{j^\star}) \tag{28}$$

TABLE 1 List of Common Patterns of Dependency Functions, Their CNOT Costs, and an Example of Their Realization as a Quantum Circuit (For Two Inputs)

Dependency function	Cost (#CNOTs)	Quantum circuit (for one and two inputs)
$g_i = l_0$	1	
$g_i = \neg l_0$	1	
$g_i = l_0 \oplus \ldots \oplus l_{m-1}$	m	
$g_i = \neg(l_0 \oplus \ldots \oplus l_{m-1})$	m	
$g_i = l_0 \cdots l_{m-1}$	2^m	
$g_i = \neg(l_0 \cdots l_{m-1})$	2^m	

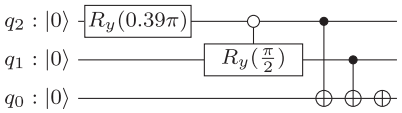


FIGURE 7. Quantum circuit to prepare f_W by UQSP$_{FD}$algorithm.

where

$$j^\star = \underset{j=0,\ldots,m-1}{\operatorname{argmax}} |t_j|. \tag{29}$$

For a given dependency function g_i in the ESOP form, we compute both $\gamma_3(g_i)$ and $\gamma_5(g_i)$, and return the minimum of them as the number of CNOTs.

Table 1 shows common cases of dependency functions, their CNOT costs, and realizations as quantum circuits by example.

Example 5: Consider f_W in Example 1. We can represent it with its minterms as

$$f_W(x_0, x_1, x_2) = \bar{x}_2\bar{x}_1x_0 + \bar{x}_2x_1\bar{x}_0 + x_2\bar{x}_1\bar{x}_0. \tag{30}$$

We derive the dependency function

$$g_0(x_1, x_2) = \neg(x_1 \oplus x_2) \tag{31}$$

with the CNOT cost

$$\gamma(g_0) = 2. \tag{32}$$

By considering the quantum circuit corresponding to the XNOR dependency function in Table 1, we get the circuit in Fig. 7 to prepare f_W using the UQSP$_{FD}$algorithm. Comparing Fig. 4 and 7, shows that we reduce the number of CNOTs by 4. Note that the 2-controlled $R_y(\pi)$ gate in Fig. 4 decomposes into 6 CNOTs.

C. VARIABLE REORDERING METHODS

Variable reordering affects both extracting dependencies and quantum state preparation, e.g., the *AND* operation is not reversible, such that variable reordering changes dependency extraction. Three variable reordering methods are evaluated to reduce the number of CNOTs.

1) *Exhaustive Reordering* (ER) preforms quantum state preparation for all permutations of the variables of the Boolean function. For each variable order, a quantum circuit is constructed. The quantum circuit with the smallest number of CNOTs is returned. Exhaustive reordering does not scale and is only practical for the Boolean function of a few Boolean variables.
2) *Random Reordering* (RR) generates prior fixed numbers of different random permutations of the variables relying on an implemented pseudorandom number generation.
3) *Greedy Reordering* (GR) generates the set of all variable orders obtained by locally swapping two variables of the function. The algorithm evaluates the number of required CNOTs to synthesize the quantum circuit under the considered orders and repeats its task using the variable order with the lowest costs as the starting point. The algorithm proceeds until a local optimum is found and the number of CNOTs cannot be improved by swapping variables.

VIII. RESULTS AND DISCUSSION

We compare the proposed UQSP using decision diagrams and dependency analysis methods with QisKit [19] which implements the method presented by Iten *et al.* [5]. Both methods are implemented in an open-source tool.[1] We specify the maximum level of optimization in QisKitto generate the circuit with the minimum number of CNOTs. All experiments are conducted on an Intel Core i7, 2.7 GHz with 16 GB memory. We show, by comparing to QisKit, that our algorithm UQSP, specialized for uniform quantum states, is more scalable and can substantially reduce the number of CNOTs over methods for arbitrary state preparation. In this section, we introduce the practical benchmarks for our experiments. Afterward, we evaluate our UQSP using decision diagrams and functional dependencies, respectively. In UQSP using functional dependencies (UQSP$_{FD}$), we examine the effect of using different dependency analysis and variable reordering methods and construct a tradeoff between them. Finally, we select our UQSP$_{FD}$as an improved algorithm and compare our results with the results extracted from QisKit for the practical benchmarks.

A. BENCHMARKS

We present experiments for a large set of uniform quantum states including the well-known quantum states GHZ and W. We also evaluate our algorithm to prepare uniform quantum states required by the *QBA* and *QCC* problems. Moreover, as we map uniform quantum states to Boolean functions and we

[1] A C++ library for quantum state preparation that will be published after notification.

TABLE 2 Experimental Results Regarding Different Dependency Analysis Methods

			Baseline [17]		Pattern search			SAT-based ESOP synthesis		
Bench	#Qs	#functions	#CNOTs	Time (s)	#CNOTs	Improvement (%)	Time (s)	#CNOTs	Improvement (%)	Time (s)
EPFL	4	367	4272	0.01	3990	6.60	0.02	3972	7.02	0.03
EPFL	5	1954	47430	0.12	45086	4.94	0.30	44714	5.73	0.42
EPFL	6	6424	289036	0.72	276424	4.36	2.62	272753	5.63	3.17
EPFL	7	14334	1208777	3.62	1087473	10.04	18.02	1065106	11.89	21.96
EPFL	8	23904	4137028	10.50	3525308	14.79	91.68	3488039	15.69	91.09
ISCAS	4	225	2358	0.01	2266	3.90	0.01	2248	4.66	0.02
ISCAS	5	676	14441	0.04	13527	6.33	0.09	13266	8.14	0.18
ISCAS	6	1150	48655	0.12	44874	7.77	0.44	43495	10.61	0.88
ISCAS	7	1452	124255	0.32	101603	18.23	1.72	97912	21.20	3.36
ISCAS	8	1571	273233	0.50	210139	23.09	4.91	203904	25.37	7.42

use some techniques from logic synthesis, we extract several Boolean functions from the EPFL and ISCAS benchmarks, which are frequently used in logic synthesis.

B. USING DECISION DIAGRAM

The advantage is that we can use $\text{UQSP}_{\text{DD}}$for preparing a larger number of qubits.

When using BDD as the function representation, we can enable a fast and scalable quantum state preparation with respect to the size of the decision diagram. Experimental results show that the proposed approach can achieve a significant reduction in runtime as compared to QisKit. By increasing the number of qubits to 30, our method prepares states in milliseconds whereas QisKitcannot prepare states for more than 18 qubits. Moreover, the results show that we can reduce the number of elementary quantum gates over QisKitby removing redundancies in the BDD. The detailed results are available in [17].

C. USING DEPENDENCY ANALYSIS

We evaluate UQSP_{FD}in terms of runtime and the number of synthesized elementary quantum gates. We primarily focus on the number of CNOTs which are more expensive than single-qubit gates for NISQ quantum computers, but we also reduce rotation gates. We use the EPFL and ISCAS benchmarks to evaluate our dependency analysis and variable reordering methods to construct a good tradeoff between them regarding runtime and the number of CNOTs. Next, we compare our final results by $\text{UQSP}_{\text{FD}}$with QisKit for the practical benchmarks such as the GHZ state, W state, and uniform quantum states required by the QBA and QCC with different k values (k-QCC).

1) DEPENDENCY ANALYSIS METHODS

We compare the general functional decomposition method, presented in Section V, against $\text{UQSP}_{\text{FD}}$with two different functional dependency methods. We implemented all methods using the same *truth table* package to represent Boolean functions. Truth tables are an effective representation for Boolean functions of up to 16 variables. For larger states with more variables, symbolic representations such as decision diagrams have to be used.

Table 2 presents the results of the general functional decomposition method as the baseline, and $\text{UQSP}_{\text{FD}}$with the proposed dependency analysis methods, pattern search, and SAT-based ESOP synthesis. The state preparation is done for one fixed variable order $n-1,\ldots,0$. For each benchmark, the number of CNOTs and the runtime in seconds are shown accumulated over all functions. The CNOT reduction increases with the number of qubits. The number of CNOTs are reduced by up to 14.79% and 15.69% for the EPFL benchmarks and by up to 23.09% and 25.37% for the ISCAS benchmarks with pattern search and SAT-based ESOP synthesis, respectively. The comparison between pattern search and SAT-based ESOP synthesis shows that the ESOP-based approach reduces the number of CNOTs further at the cost of requiring more runtime. Selecting the strategy allows users to trade runtime for quality of results depending on the application scenario.

2) VARIABLE REORDERING METHODS

We further examine the impact of the proposed variable reordering methods, ER which considers all $n!$ orders, RR with n^2 different randomly chosen orders using a fixed random seed, and GR which dynamically reorders until no further local improvement is achieved, considering SAT-based ESOP as the dependency strategy.

The experimental results using ESOP-based dependency analysis with ER, RR, and GR are summarized in Table 3. For each reordering method, we show the number of CNOTs, the relative improvement over the baseline, and the required runtime in seconds. Exhaustive reordering reaches the best result, but requires too much runtime and can only be applied to small functions in practice. Comparison of RR and GR shows that the CNOTs are reduced using RR and runtime is increased because the number of considered orders using RR are more than GR. But as CNOTs reduction is more important for us and runtime consumption using RR is still less, we select RR over GR. As a result, to set the best tradeoff between the number of CNOTs and runtime, for a small number of qubits, we do preparation using ER and for a large number, using RR.

3) COMPARISON TO QISKITFOR THE PRACTICAL QUANTUM STATES

The experimental results for the quantum state preparation of the practical quantum states are presented in Table 4. As benchmarks, the Boolean functions for the GHZ and W

TABLE 3 Experimental Results Regarding Different Variable Reordering Methods

Bench	#Qs	#functions	ER with $n!$ orders			RR with n^2 orders			GR		
			#CNOTs	Improvement (%)	Time (s)	#CNOTs	Improvement (%)	Time (s)	#CNOTs	Improvement (%)	Time (s)
EPFL	4	367	3646	14.65	0.44	3686	13.72	0.25	3938	7.82	0.11
EPFL	5	1954	39509	16.70	24.03	39895	15.89	3.89	44247	6.71	1.91
EPFL	6	6224	232679	19.50	953.33	237048	17.99	45.46	270673	6.35	16.73
EPFL	7	14334	730474	39.57	36257.95	790067	34.64	325.38	1035187	14.36	136.11
ISCAS	4	225	2064	12.47	0.30	2080	11.79	0.17	2238	5.09	0.07
ISCAS	5	676	12043	16.61	10.13	12104	16.18	1.49	13248	8.26	0.78
ISCAS	6	1150	38828	20.20	233.99	39294	19.24	10.83	43226	11.16	4.48
ISCAS	7	1452	72043	42.02	5170.97	76162	38.71	42.55	94305	24.10	22.41

TABLE 4 UQSP_{FD} Results in Comparison to QisKit Results for the Practical Quantum States

Bench	#Qs	UQSP_{FD} (ESOP+RR)			QisKit			Improvement (%)		
		#CNOTs	#Rot. gates	Time (s)	#CNOTs	#Rot. gates	Time (s)	#CNOTs	#Rot. gates	Time (s)
GHZ	10	9	1	0.00	1013	1023	6.36	99.11	99.90	100.00
GHZ	12	11	1	0.00	4083	4094	25.63	99.73	99.98	100.00
GHZ	15	14	1	0.00	32752	32767	246.81	99.96	100.00	100.00
W	10	519	512	0.01	1013	1023	7.09	48.77	49.95	99.86
W	12	2057	2048	0.01	4083	4095	28.73	49.62	49.99	99.96
W	15	16396	16384	0.01	32752	32767	244.91	49.94	50.00	99.99
QBA	12	289	289	7.56	435	515	28.21	33.56	43.88	73.20
QBA	15	1438	1439	22.52	32620	32632	455.31	95.59	95.59	95.05
QBA	16	0	12	44.58	180	302	518.01	100.00	96.03	91.39
QBA	18	2343	2343	345.41	114643	114656	4138.01	97.96	97.96	91.65
64_QCC	8	127	128	0.15	247	255	2.25	48.58	49.80	93.33
3996_QCC	12	4094	4095	5.45	4063	4069	48.96	-0.76	-0.64	88.87
1024_QCC	14	16382	16383	13.14	16367	16380	205.62	-0.09	-0.02	93.61
256_QCC	16	32766	32768	40.7	65365	65248	912.32	49.87	49.78	95.54
total		76445	76404	479.54	309616	309826	6868.22	75.31	75.34	93.02

states as well as QBA and k-QCC are considered, over n qubits (#Qs). We consider the Boolean functions for different numbers of qubits in the range 8–18 qubits. Our approach improves the number of CNOTs, rotation gates, and runtime over QisKit. The proposed method converges to the optimum circuit for the GHZ state, whereas QisKituses a fixed precomputed optimal template for this state. Note that in Table 4, QisKit's results come from applying their algorithm, not using a template. For the W state, our approach reduces CNOTs significantly whereas QisKit's results are close to the upper bound of $2^n - 2$.

The QBA benchmark consists of a sequence of 1-bits in the beginning and a sequence of 0-bits for the rest in its truth table. This means we can divide qubits into three parts. We have all input assignments for the first part of qubits that we can prepare them using only rotation gates without any control qubits. For the second part, we need control qubits to prepare them as we do not have all input assignments for them but we can utilize dependencies to reduce control qubits. For the third part, we do not have any input assignment so these qubits will be in zero state. Results in Table 4 show that our method can deal better over QisKit for this benchmark. For example, our method prepares QBA with 16 qubits [QBA(16)] only with 12 rotation gates whereas QisKit requires 180 CNOTs. From (15), QBA(16) consists of 2^{12} 1-bits that shows we have all input assignments for the first 12 qubits. This shows that our method prepares QBA state efficiently.

For the k-QCC benchmark, we select four different benchmarks with different k values randomly. As shown in the table, for some cases we reduce the number of CNOTs by exploiting dependencies between qubits. But for some other cases, for example 3996-QCC and 1024-QCC, there is not any dependencies between qubits and our results are close to the upper bound whereas the QisKit's results are a little bit better. This is due to the fact that QisKit uses post-optimization methods whereas we do not apply any postoptimization.

In total, UQSP_{FD} reduces CNOTs, rotation gates, and runtime for quantum states of up to 18 qubits in average by 75.31%, 75.34%, and 93.02%, respectively.

IX. CONCLUSION

To address the general problem of preparing quantum states, a Boolean method specialized for preparing uniform quantum states is proposed, called UQSP. Uniform quantum states are an important family of states that are frequently used in quantum algorithms, e.g., quantum cryptography. Our method uses Shannon's decomposition and cofactoring. We implemented UQSP using both truth table and BDD representations as we can efficiently apply Shannon's decomposition to them. Using BDDs helps us to reduce runtime and prepare quantum states with larger number of qubits where state-of-the-art methods are not applicable.

To reduce the number of CNOTs, we identify functional dependencies among qubits and use them to construct optimized quantum circuits. We added functional dependencies as an extension to our UQSP, called UQSP_{FD}. We implemented UQSP_{FD} using truth tables. The experimental results show that preparing uniform quantum states using this

specialized method reduces both CNOTs and runtime over QisKit as the state-of-the-art method. These results clearly indicate that specialized procedures for preparing quantum states improve over general methods for preparing arbitrary quantum states. As future works, we will focus on identifying other families of quantum states that can be efficiently prepared.

ACKNOWLEDGMENT

The authors would like to thank S.-Y. Lee for giving feedback on the manuscript.

REFERENCES

[1] M. Mottonen, J. J. Vartiainen, V. Bergholm, and M. M. Salomaa, "Transformation of quantum states using uniformly controlled rotations," *Quantum Inf. Comput.*, vol. 5, no. 6, pp. 467–473, 2005, doi: 10.26421/QIC5.6-5.

[2] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 1000–1010, Jun. 2006, doi: 10.1109/TCAD.2005.855930.

[3] P. Kaye and M. Mosca, "Quantum networks for generating arbitrary quantum states," in *Proc. Int. Conf. Quantum Inf.*, 2001, Art. no. PB 28, doi: 10.1364/ICQI.2001.PB28.

[4] P. Niemann, R. Datta, and R. Wille, "Logic synthesis for quantum state generation," in *Proc. IEEE 46th Int. Symp. Multiple-Valued Log.*, 2016, pp. 247–252, doi: 10.1109/ISMVL.2016.30.

[5] R. Iten, R. Colbeck, I. Kukuljan, J. Home, and M. Christandl, "Quantum circuits for isometries," *Phys. Rev. A*, vol. 93, no. 3, 2016, Art. no. 032318, doi: 10.1103/PhysRevA.93.032318.

[6] A. Zulehner, S. Hillmich, I. L. Markov, and R. Wille, "Approximation of quantum states using decision diagrams," in *Proc. 25th Asia South Pacific Des. Automat. Conf.*, 2020, pp. 121–126, doi: 10.1109/ASP-DAC47756.2020.9045454.

[7] C. Zoufal, A. Lucchi, and S. Woerner, "Quantum generative adversarial networks for learning and loading random distributions," *NPJ Quantum Inf.*, vol. 5, no. 1, 2019, Art. no. 103, doi: 10.1038/s41534-019-0223-2.

[8] I. F. Araujo, D. K. Park, F. Petruccione, and A. J. da Silva, "A divide-and-conquer algorithm for quantum state preparation," *Sci. Rep.*, vol. 11, 2020, Art. no. 6329, doi: 10.1038/s41598-021-85474-1.

[9] M. Plesch and Č. Brukner, "Quantum-state preparation with universal gate decompositions," *Phys. Rev. A*, vol. 83, no. 3, 2011, Art. no. 032302, doi: 10.1103/PhysRevA.83.032302.

[10] S. Arunachalam, A. Belovs, A. M. Childs, R. Kothari, A. Rosmanis, and R. de Wolf, "Quantum coupon collector," in *Proc. 15th Conf. Theory Quantum Comput., Commun. Cryptography. Schloss Dagstuhl-Leibniz-Zentrum für Informatik*, 2020, pp. 10:1–10:17, doi: 10.4230/LIPIcs.TQC.2020.10.

[11] D. M. Greenberger, M. A. Horne, and A. Zeilinger, "Going beyond Bell's theorem," in *Bell's Theorem, Quantum Theory and Conceptions of the Universe*. Berlin, Germany: Springer, 1989, pp. 69–72, 10.1007/978-94-017-0849-4_10.

[12] W. Dür, G. Vidal, and J. I. Cirac, "Three Qubits can be entangled in Two inequivalent ways," *Phys. Rev. A*, vol. 62, no. 6, 2000, Art. no. 062314, doi: 10.1103/PhysRevA.62.062314.

[13] N. Shenvi, J. Kempe, and K. B. Whaley, "Quantum random-walk search algorithm," *Phys. Rev. A*, vol. 67, no. 5, 2003, Art. no. 052307, doi: 10.1103/PhysRevA.67.052307.

[14] M. Ben-Or and A. Hassidim, "Fast quantum Byzantine agreement," in *Proc. 37th Annu. ACM Symp. Theory Comput.*, 2005, pp. 481–485, doi: 10.1145/1060590.1060662.

[15] M. Hillery, V. Bužek, and A. Berthiaume, "Quantum secret sharing," *Phys. Rev. A*, vol. 59, no. 3, 1999, Art. no. 1829, doi: 10.1103/PhysRevA.59.1829.

[16] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York, NY, USA: McGraw-Hill, 1994, doi: 10.5860/choice.32-0950.

[17] F. Mozafari, M. Soeken, H. Riener, and G. De Micheli, "Automatic uniform quantum state preparation using decision diagrams," in *Proc. IEEE 50th Int. Symp. Mult.-Valued Log.*, 2020, pp. 170–175, doi: 10.1109/ISMVL49045.2020.00-10.

[18] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. 100, no. 8, pp. 677–691, Aug. 1986, doi: 10.1109/TC.1986.1676819.

[19] G. Aleksandrowicz *et al.*, "Qiskit: An open-source framework for quantum computing," 2019, doi: 10.5281/zenodo.2562110.

[20] R. Babbush *et al.*, "Encoding electronic spectra in quantum circuits with linear t complexity," *Phys. Rev. X*, vol. 8, no. 4, 2018, Art. no. 041015, doi: 10.1103/PhysRevX.8.041015.

[21] L. Grover and T. Rudolph, "Creating superpositions that correspond to efficiently integrable probability distributions," 2002, *arXiv:quant-ph/0208112*.

[22] T. Sasao, "An exact minimization of AND-EXOR expressions using reduced covering functions," in *Proc. Synth. Simul. Meeting Int. Interchange*, 1993, pp. 374–383.

[23] H. Riener, R. Ehlers, B. deO.Schmitt, and G. De Micheli, *Exact Synthesis of ESOP Forms*. Cham, Switzerland: Springer, 2020, pp. 177–194, doi: 10.1007/978-3-030-20323-8.

[24] A. Mishchenko and M. Perkowski, "Fast heuristic minimization of exclusive-sums-of-products," in *Proc. Int. Workshop Appl. Reed–Muller Expansion Circuit Des.*, 2001, pp. 242–250.

[25] M. Soeken, G. Meuli, B. Schmitt, F. Mozafari, H. Riener, and G. De Micheli, "Boolean satisfiability in quantum compilation," *Philos. Trans. Roy. Soc. A*, vol. 378, no. 2164, 2020, Art. no. 20190161, doi: 10.1098/rsta.2019.0161.

[26] G. Meuli, B. Schmitt, R. Ehlers, H. Riener, and G. De Micheli, "Evaluating ESOP optimization methods in quantum compilation flows," in *Proc. Int. Conf. Reversible Comput.*, 2019, pp. 191–206, doi: 10.1007/978-3-030-21500-2.

[27] S. B. Akers, "Binary decision diagrams," *IEEE Comput. Architecture Lett.*, vol. C-27, no. 06, pp. 509–516, Jun. 1978, doi: 10.1109/TC.1978.1675141.

[28] R. P. Feynman, "Quantum mechanical computers," *Foundations Phys.*, vol. 16, no. 6, pp. 507–531, 1986, doi: 10.1007/BF01886518.

[29] M. Nielsen and I. Chuang, *Quantum Computation and Quantum Information*. Cambridge, U.K.: Cambridge Univ. Press, 2000, doi: 10.1017/CBO9780511976667.

[30] V. Bergholm, J. J. Vartiainen, M. Möttönen, and M. M. Salomaa, "Quantum circuits with uniformly controlled one-qubit gates," *Phys. Rev. A*, vol. 71, no. 5, 2005, Art. no. 0 52330, doi: 10.1103/PhysRevA.71.052330.

[31] Wikipedia, "Quantum Byzantine agreement," Accessed on: Nov. 18, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Quantum_Byzantine_agreement

[32] B. Schmitt, F. Mozafari, G. Meuli, H. Riener, and G. De Micheli, "From Boolean functions to quantum circuits: A scalable quantum compilation flow in C++," in *Proc. Des., Automat. Test Eur. Conf. Exhib.*, 2021, pp. 1044–1049, doi: 10.23919/DATE51398.2021.9474237.

[33] J.-H. R. Jiang, and R. K. Brayton, "Functional dependency for verification reduction," in *Proc. Int. Conf. Comput. Aided Verification Proc. Int. Conf. Comput. Aided Verification*, 2004, pp. 268–280, doi: 10.1007/978-3-540-27813-9.

[34] N. Schuch and J. Siewert, "Programmable networks for quantum algorithms," *Phys. Rev. Lett.*, vol. 91, no. 2, 2003, Art. no. 027902, doi: 10.1103/PhysRevLett.91.027902.

[35] J. Welch, D. Greenbaum, S. Mostame, and A. Aspuru-Guzik, "Efficient quantum circuits for diagonal unitaries without Ancillas," *New J. Phys.*, vol. 16, no. 3, 2014, Art. no. 033040, doi: 10.1088/1367-2630/16/3/033040.