Configurable sublinear circuits for quantum state preparation

Israel F. Araujo1,2,3 · Daniel K. Park2,3 · Teresa B. Ludermir1 · Wilson R. Oliveira4 · Francesco Petruccione5,6,7 · Adenilton J. da Silva1

Abstract
The theory of quantum algorithms promises unprecedented benefits of harnessing the laws of quantum mechanics for solving certain computational problems. A prerequisite for applying quantum algorithms to a wide range of real-world problems is loading classical data to a quantum state. Several circuit-based methods have been proposed for encoding classical data as probability amplitudes of a quantum state. However, in these methods, either quantum circuit depth or width must grow linearly with the data size, nullifying the advantage of representing exponentially many classical data in a quantum state. In this paper, we present a configurable bidirectional procedure that addresses this problem by tailoring the resource trade-off between quantum circuit width and depth. In particular, we show a configuration that encodes an N-dimensional classical data using a quantum circuit whose width and depth both grow sublinearly with N. We demonstrate proof-of-principle implementations on five quantum computers accessed through the IBM and IonQ quantum cloud services.

Keywords Quantum computing · State preparation · Bidirectional · Circuit optimization

1 Introduction
In 1981, Richard Feynman suggested a computational architecture based on cellular automata with quantum features and conjectured that it could accelerate simulations of quantum physics and chemistry beyond the capability of classical computers [1]. Since then, algorithms such as Deutsch–Jozsa [2], Grover [3], Simon [4], and Shor [5] demonstrated quantum advantages. However, many quantum algorithms face theoretical challenges that prevent quantum advantages [6–8]. Among these challenges
is the development of an efficient encoding of complex classical data into a quantum state [6–9].

Quantum algorithms assume an initial quantum state prepared before the computation. The worst case complexity of preparing an arbitrary quantum state is exponential with the number of qubits [10]. For this reason, the most significant quantum speed-ups occur when the quantum algorithm [2–5, 11–13] operates on an input state that is easy to prepare, such as the uniform superposition of all computational basis states. For algorithms that rely on loading data into an arbitrary quantum state, an efficient means to prepare input states is a prerequisite to quantum speed-ups [6, 7, 9, 14].

While the quantum state preparation models based on quantum oracles [15–17] or quantum random access memory [18–25] are useful for evaluating the lower bounds of the computational cost and identifying the complexity class, implementations of them must be considered in practice. In particular, the quantum speed-up can vanish without an efficient implementation of quantum state preparation when quantum algorithms carry classical data in a non-uniform quantum superposition. Examples of such instances include Quantum Machine Learning (QML) [6, 7, 26–32], Quantum Memories (QMem) [18–25], and Quantum Linear Algebra (QLA) [7, 14, 33–36]. Quantum machine learning algorithms try to estimate a target function from a finite set of example points by unveiling correlations between inputs and outputs of the correspondent function [7, 29, 37]. Quantum memories must store a set of samples from a configuration space as a superposition state before the information is retrieved using the algorithm [18]. Quantum linear algebra algorithms operate with a critical assumption that classical data have been efficiently encoded as probability amplitudes of a quantum state, without which the quantum speed-up vanishes [6, 7, 9, 14]. All of the above emphasize the importance of developing efficient quantum state preparation algorithms for broad application of quantum computing techniques on classical data.

Several solutions to the problem of quantum state preparation have been proposed [10, 15, 38–43], but all produce circuits with width or depth growing at least linearly with the size of the input vector [10]. For example, the top-down method proposed in Ref. [38] achieves the exponential compression of the quantum circuit width while requiring $O(N)$ quantum circuit depth for $N$-dimensional data ($N = 2^n$). On the other extreme end, the bottom-up method [43] achieves the exponential compression of the quantum circuit depth while requiring $O(N)$ quantum circuit width and entangled information in ancillary qubits. Since there is an extra resource overhead in many quantum algorithms due to the quantum measurement postulate [24, 44], such linear cost can impose restrictions on possible speed-ups, dominating the computational cost of the intended quantum application. Other approaches have reduced circuit complexity to initialize an approximate quantum state [15, 45–47], but this paper targets the exact state preparation with entangled ancillary qubits.

This work presents a quantum state preparation method that achieves sublinear scaling on both quantum circuit resources. More specifically, a bidirectional strategy that effectively combines the aforementioned approaches in a way that the trade-off between computational time and space can be configured. Both temporal and spatial complexities depend on the parameter $s \in [1 \ldots n]$, which adjusts the trade-off between computational time and space. Given an $N$-dimensional input vector, the total time complexity of the bidirectional algorithm is $O_c(N) + O_d(2^s + \log_2^2(N))$,
where $O_c(N)$ is the time of the classical preprocessing to create the quantum circuit and $O_d(2^s + \log_2^2(N))$ is the quantum circuit depth. Typically, the same input vector is loaded $l \gg N$ times, and hence, the amortized computational time is $O_d(2^s + \log_2^2(N))$. Note that classical preprocessing is also common in classical computing and is necessary in other quantum state preparation methods as well. The spatial complexity (i.e., the width) of the circuit is $O_w(sN/2^s)$.

Besides the sublinear circuit cost, the ability to customize the exchange between these quantum resources is advantageous when realistic quantum hardware specifications are considered, as one resource can be cheaper than the other to scale up. For instance, it is a useful feature for future Noisy Intermediate-Scale Quantum (NISQ) devices with the promise of computers with a large number of physical qubits [48], albeit noise limits the depth of the circuits [49].

Decreasing the depth of the circuit is achieved by increasing the circuit width and creating entanglement between the data register qubits and an ancillary system. Thus, when the data register is considered alone by tracing out the ancillary qubits, the resulting state is mixed, which can be a disadvantage in the case of quantum applications for solving linear systems [14] or differential equations [50]. However, the classical data are still encoded as probability amplitudes of an orthonormal basis set. Useful applications, such as supervised quantum machine learning and statistical analysis, can be constructed based on this [43].

This paper is divided into four sections. Section 2 reviews two strategies for loading classical information into quantum devices, namely top-down [40] and bottom-up [43] approaches. The former is used by quantum computing libraries [51, 52] as the method for general quantum amplitude initialization. These methods are at the two opposite ends of the quantum circuit cost spectrum, requiring either the maximal circuit depth or width to minimize the other resource. Section 3 presents the main result, a bidirectional method that combines the top-down and bottom-up strategies in a configurable manner. Complexity expressions for the bidirectional method are established in Sect. 3.1, which shows that the bottom-up and the top-down strategies are recovered when $s = 1$ and $s = n$, respectively, and that sublinear scaling on both depth and width is possible when $s = n/2$. Proof-of-principle experiments performed on superconducting and ion-trap based quantum devices are presented in Sect. 3.2. Section 4 presents the conclusion and perspectives for future work.

2 Quantum state preparation with linear cost

2.1 Tree representation

Quantum state preparation algorithms aim to load classical data into a quantum computer, and there are different ways to encode such information as a quantum state [29, 53]. One relevant strategy is called amplitude encoding, which creates a state $\sum_p |x_p\rangle e^{i\omega_p} |p\rangle$ that encodes a normalized vector $x = (|x_0\rangle e^{i\omega_0}, \ldots, |x_{N-1}\rangle e^{i\omega_{N-1}})$ as the probability amplitudes. Several of the existing methods for amplitude encoding can be understood as a walk on a binary tree [10, 40, 42, 43, 54]. Each tree node corresponds to a controlled gate operation, and the height increases with the number
of qubits (see Fig. 1a, b). Two edges stemming from each node indicate that each controlled gate operation splits the Hilbert space into two subspaces. Therefore, after $n$ layers, there can be $2^n$ subspaces with distinct probability amplitudes. Depending on the choice of the walk direction, different state preparation strategies, such as top-down and bottom-up approaches, can be constructed.

To explain the procedure, four parameters [40] defined by the target vector $\mathbf{x}$ are introduced as

$$\Omega_{i,k} = \sum_{l=0}^{2^k-1} \omega_{(i-1)2^k+l}/2^{k-1}$$

$$\eta_{i,k} = \sqrt{\sum_{l=0}^{2^k-1} |x_{(i-1)2^k+l}|^2}$$

$$\lambda_{j,v} = \Omega_{2j,v-1} - \Omega_{j,v}$$

$$\beta_{j,v} = \eta_{2j,v-1}/\eta_{j,v}$$

where $j = 1, 2, \ldots, 2^{n-v}$, $v = 1, 2, \ldots, n$, and $n = \log_2(N)$. These parameters are used to construct the tree representations of the state preparation algorithms, namely the state tree (Fig. 1a) and the angle tree (Fig. 1b). Indices $k$ and $v$ indicate a tree level in ascending order from the leaf nodes to the root, and $i$ and $j$ are node indices at a given level. The nodes of these trees are complex values that represent the amplitudes of the quantum state to be encoded and the rotation angles for the construction of the encoding quantum circuit. The magnitude and complex argument of the state tree amplitudes are obtained through $\eta_{i,k}$ and $\Omega_{i,k}$, respectively. When $k = 0$, the parameters point to the input vector $\mathbf{x}$, with $\eta_{i,0} = |x_{i-1}|$ and $\Omega_{i,0} = 2\omega_{i-1}$. Equations (3) and (4) determine rotation values of the angle tree nodes. The phase arguments of the vector $|\mathbf{x}\rangle$ are encoded through z-rotations of angles $\lambda_{j,v}$, and the magnitudes through y-rotations of angles $\alpha_{j,v} = 2\sin(\beta_{j,v})$.

Algorithms 1 and 2 describe the construction of a state tree and an angle tree. Respective pseudocodes 1 and 2 are presented in the “Appendix”.

Fig. 1 Tree representations of quantum state preparation algorithms. a State decomposition tree generated by Algorithm 1 with an eight-dimensional input vector $\mathbf{x}$ (dashed nodes). The complex argument terms $\Omega_{i,k}$ were omitted for readability. b Angle tree generated by Algorithm 2 with an eight-dimensional input vector. The correspondent phase angles $\lambda_{j,v}$ were omitted for readability.
Algorithm 1: State tree construction
1. Initialize the state tree by the leaves, where each node value is a complex amplitude from a $2^n$ length state vector.
2. Set $k = 1$
3. Create a new level with $2^{n-k}$ nodes, where each node $i$ value is $\eta_i e^{i \Omega_{i,k}}$ (Eq. (1) and Eq. (2)), $i = 1, \ldots, 2^{n-k}$.
4. If $k < n$, set $k = k + 1$ and return to Step 2, otherwise output the state tree.

Algorithm 2: Angle tree construction
1. Set $v = n$.
2. Create a new level $v$ with $2^{n-v}$ nodes, where each node $j$ stores the angles $\alpha_{j,v}$ and $\lambda_{j,v}$ (Eq. (4) and Eq. (3)), $\alpha_{j,v} = 2 \sin(\beta_{j,v})$, $j = 1, \ldots, 2^{n-v}$, using data from a state tree generated by Algorithm 1.
3. If $v > 1$, set $v = v - 1$ and return to Step 1, otherwise output the angle tree.

2.2 Top-down approach

The top-down amplitude encoding approach to quantum state initialization is a linear transformation consisting of a sequence of uniformly controlled rotations [40, 54] that takes the initial basis vector $|0\rangle^\otimes n$ to some arbitrary vector $|x\rangle = (|x_0 e^{i \omega_0}, \ldots, |x_{N-1} e^{i \omega_{N-1}} \rangle)^T$. This generates a quantum circuit with complexity of $O_d(N)$ and $O_w(\log_2(N))$ [10, 40, 54].

The top-down state preparation (TDSP) algorithm begins by preparing the following state at the root ($v = n$) of the angle tree (see Fig. 1b for an example)

$$|\psi_n\rangle = e^{-i \frac{\lambda_1}{2} \sqrt{1 - |\beta_1,n|^2}} |0\rangle + e^{i \frac{\lambda_1}{2} \beta_1,n} |1\rangle.$$  (5)

To load states into the next level (indicated by $v$ in Eq. (6)), the current state (indicated by $v+1$ because $v$ is in reverse order, decreasing from $n$ to 1) is sequentially combined with the values of the next state in Eq. (6).

$$|\psi_v\rangle = \sum_{j=1}^{2^{n-v}} |j-1\rangle \langle j-1| |\psi_{v+1}\rangle \left( e^{-i \frac{\lambda_{j,v}}{2} \sqrt{1 - |\beta_{j,v}|^2}} |0\rangle + e^{i \frac{\lambda_{j,v}}{2} \beta_{j,v}} |1\rangle \right).$$  (6)

The update of state $|\psi_v\rangle$ is repeated for $v = (n-1), \ldots, 1$, thereby obtaining the desired state

$$|\psi_1\rangle = |x_0 e^{i \omega_0}|0\rangle + \ldots + |x_{N-1} e^{i \omega_{N-1}}|N-1\rangle.$$  

The summation in Eq. (6) expresses the sequential characteristic of the top-down approach, since the state of each layer of the tree needs to be loaded on one qubit through a sequence of rotations. Figure 2 presents an example quantum circuit for encoding an eight-dimensional vector using the top-down state preparation method.
Algorithm 3: Top-down state preparation

1. Generate a state tree from the input vector
2. Generate an angle tree from the state tree
3. Create a quantum circuit with $n$ qubits (one qubit for each angle tree level)
4. Perform one y-rotation and one z-rotation on the first qubit (qubits are 0-indexed) using the angle tree root values $\alpha_{1,n}$ and $\lambda_{1,v}$ (Eq. (5))
5. Set $v = n − 1$ (starts at the root)
6. Perform a Uniformly Controlled Rotation controlled by qubits $0, 1, \ldots, n − v − 1$ (corresponding to the previous levels) with the current qubit $n − v$ as target, using the current level nodes values $\alpha_{j,v}$ and $\lambda_{j,v}$ ($1 \leq j \leq 2^{n-v}$) as rotation angles (Eq. (6))
7. If $v > 1$, set $v = v − 1$ and return to Step 5, otherwise output the encoding quantum circuit

The name top-down comes from the way this approach walks through the tree from the root to the leaves to build a quantum circuit. The combination of states is done with multi-controlled rotations, and it takes $\log_2(N)$ qubits to generate the complete state. At each level, it assembles a sequence of rotations targeting one qubit and is controlled by the qubits of the previous levels. First, y-rotations are applied to set the magnitudes, followed by z-rotations to set the phases. These steps are presented in Algorithm 3 with its Pseudocode 3 provided in the “Appendix”.

2.3 Bottom-up approach

The bottom-up state preparation algorithm, also known as divide-and-conquer quantum state preparation [43, 53], constructs a quantum circuit with complexity $O_d(\log^2(N))$ and $O_w(N)$. It starts by preparing $N/2$ single-qubit states, corresponding to the leaves of the tree (Fig. 1a). Equations (3) and (4) are used starting from the lowest level of the tree ($v = 1$), which corresponds to starting from the initial state

$$|\psi_{j,1}\rangle = e^{-i\frac{\lambda_{j,1}}{2}} \sqrt{1-|\beta_{j,1}|^2} |0\rangle + e^{i\frac{\lambda_{j,1}}{2}} \beta_{j,1} |1\rangle.$$  

Loading the states in the upper levels of the tree is done by recursive updates of

$$|\psi_{j,v}\rangle = e^{-i\frac{\lambda_{j,v}}{2}} \sqrt{1-|\beta_{j,v}|^2} |0\rangle |\psi_{j-1,v-1}\rangle |\psi_{j,v-1}\rangle + \\ e^{i\frac{\lambda_{j,v}}{2}} \beta_{j,v} |1\rangle |\psi_{j-1,v-1}\rangle |\psi_{j,v-1}\rangle.$$  


Fig. 3 Divide-and-conquer bottom-up load strategy. a Circuit generated by the divide-and-conquer [43] bottom-up strategy (Algorithm 4) to load an 8-dimensional complex vector in a quantum device. The indexes of the qubits correspond to the tree nodes indexes in Fig. 1b. The circuit starts with the simultaneous preparation of \((N - 1)\) one-qubit states associated with all tree nodes, followed by the combination of states through CSWAPs. b Combining states with controlled-swap operations

where \(v = 2, \ldots, n\). The desired state, with ancilla \(|\phi\rangle\), is obtained when \(v = n\) as

\[
|\psi_{1,n}\rangle = |x_0| e^{i\alpha_0} |0\rangle|\phi_0\rangle + \cdots + |x_{N-1}| e^{i\alpha_{N-1}} |N - 1\rangle|\phi_{N-1}\rangle.
\] (9)

**Algorithm 4:** Bottom-up state preparation

1. Generate a state tree from the input vector
2. Generate an angle tree from the state tree
3. Create a quantum circuit with \(2^n - 1\) qubits (one qubit for each angle tree node)
4. Perform \(2^{n-1}\) y-rotations and z-rotations on qubits \(2^{n-1} + j - 2\) \((1 \leq j \leq 2^{n-1})\) using the leaf values \(\alpha_{j,1}\) and \(\lambda_{j,1}\) to prepare \(2^{n-1}\) initial single-qubit states (Eq. (7), Fig. 3a)
5. Set \(v = 2\) and \(j = 1\) (starts at the bottom)
6. Perform one y-rotation and one z-rotation on qubit \(2^{n-v} + j - 2\) using the node values \(\alpha_{j,v}\) and \(\lambda_{j,v}\) to prepare a single-qubit state to control CSWAPs operations
7. Perform Controlled SWAPs controlled by qubit \(2^{n-v} + j - 2\) to combine the previous states prepared with the qubits associated to the sub-tree started by the current node (Eq. (8), Fig. 3b)
8. If \(j < 2^{n-v}\), set \(j = j + 1\) and return to Step 5, otherwise continue
9. If \(v < n\), set \(v = v + 1\) and return to Step 5, otherwise output the encoding quantum circuit

Updating the states in Eq. (8) requires a method that entangles each of the two states \(|\psi_{2,j-1,v-1}\rangle\) and \(|\phi_{2,j,v-1}\rangle\) to orthonormal subspaces \(|0\rangle\) and \(|1\rangle\), respectively, with designated amplitudes. As demonstrated by Araujo et al. [43], \(m\) controlled-swap (CSWAP) operations can combine two \(m\)-qubit states in the form of Eq. (8) (see Fig. 3b) to encode the desired set of amplitudes in the orthonormal subspaces of the first \(m + 1\) qubits. Since each node of the level is represented by one qubit, multiple loading within a layer can be performed in parallel. Thus, all states in the given layer can be loaded simultaneously. This is an advantage in comparison with the top-down
approach, which loads each node state sequentially. Since the underlying idea of the bottom-up approach is recursive combination of single-qubit states that are easy to prepare, it was named as divide-and-conquer state preparation (DCSP) when first introduced [43]. An example quantum circuit for encoding eight-dimensional vector using the DCSP method is depicted in Fig. 3a. Algorithm 4 describes these steps, and Pseudocode 4 is provided in the “Appendix”.

3 Bidirectional quantum state preparation

This section presents a bidirectional state preparation (BDSP) method combining both bottom-up and top-down strategies as walking on the tree in both directions. This new strategy can interchange depth and space cost in a configurable manner, thereby allowing for the sublinear cost in both quantum circuit depth and width. In particular, the equilibrium point between these costs achieves the quadratic reduction in both space and time. The algorithm is depicted in Fig. 4, and the detailed explanation is provided as follows.

The bidirectional state preparation algorithm starts by informing a level \( v = s \) (enumerated from bottom to top, where \( 1 \leq s \leq n \)) at which the angle tree is split, followed by two stages. In the first stage, it segments the tree section below \( s \) into \( 2^{n-s} \) sub-trees of height \( s \). The \( 2^{n-s} \) nodes at level \( s \) are the roots of these sub-trees. The number of sub-trees determines how many initial sub-states should be prepared in the first stage of the algorithm. The amplitude values of these sub-states \( a_j = (a_{j,1}, \ldots, a_{j,2^s}) \) \((1 \leq j \leq 2^{n-s})\) are loaded concurrently using a sequential algorithm [10, 40, 54] based on the TDSP method as

\[
|\psi_{j,s}\rangle = \sum_{k=1}^{2^s} a_{j,k} |k-1\rangle; \quad j = 1, 2, \ldots, 2^{n-s}.
\] (10)

The initial sub-states are the input of the second stage of BDSP. They reproduce the state that would be created by the bottom-up steps up to the split level \( s \). In the second stage, the sub-states are combined to generate the complete state by the divide-and-conquer approach (Fig. 4c). The bottom-up algorithm takes the state prepared in the first stage as the input and starts walking on the tree from the split level (Eq. (8), where \( v = s + 1, \ldots, n \)). In other words, the BDSP follows the bottom-up DCSP algorithm starting from states \( |\psi_{j,s}\rangle \) (Eq. (10)) instead of starting from the single-qubit leaf states (Eq. (7)). The BDSP algorithm is described in Algorithm 5 below, with Pseudocode 5 provided in the “Appendix”.

3.1 Complexity

Theorem 1 Algorithm 5 generates a quantum circuit with depth \( O_d \left( 2^s + \log_2^2 (N) \right) \) and width \( O_w \left( s \frac{N}{2^s} \right) \).
Algorithm 5: Bidirectional state preparation

1. Generate a state tree from the input vector
2. Generate an angle tree from the state tree
3. Create a quantum circuit with $(s + 1)2^{n-s} - 1$ qubits (Eq. (12))
4. Perform Algorithm 3 (top-down approach) starting from step 3 to prepare $2^{n-s}$ states of $s$ qubits (replacing $n$ by $s$), using the $2^{n-s}$ sub-trees as input for each state (Eq. (10)). This step is named Stage 1
5. Perform Algorithm 4 (bottom-up approach) starting from step 5 and $v = s + 1$ to combine the $2^{n-s}$ states prepared in Stage 1 using the remaining $2^{n-s} - 1$ qubits (Eq. (8)). This step is named Stage 2
6. Output the encoding quantum circuit
Table 1  Bidirectional quantum circuit complexity for different configurations

|                  | General $s$          | Bottom-up $s = 1$ | Top-down $s = n$ | Sublinear $s = n/2$ |
|------------------|----------------------|-------------------|------------------|---------------------|
| $O_d$            | $2^s + \log_2^2(N)$  | $n^2$             | $2^n$            | $2^{n/2}$           |
| $O_w$            | $s(N/2^s)$           | $2^s$             | $n$              | $n(2^{n/2})$        |

These expressions were obtained from Eqs. (11) and (12).

**Proof** The BDSP algorithm builds quantum circuits whose depth and width are expressed, respectively, by

$$\frac{N}{2\log_2(N) - s} + \sum_{i=\log_2(N)}^{\log_2(N)} (i - 1) = 2^s + \frac{1}{2}(\log_2^2(N) - \log_2(N) - s^2 + s)$$  

(11)

and

$$\frac{sN}{2^s} + \frac{N}{2^s} - 1 = (s + 1)\frac{N}{2^s} - 1,$$  

(12)

where $N$ is the number of amplitudes (i.e., the dimension of the data vector) and $s$ is a parameter indicating the tree splitting level (the tree level in reverse order). Stage 1 and 2 indicate the contribution from each stage of the bidirectional procedure to the circuit complexity stated in Theorem 1.

In Eq. (11), the first term (stage 1) is the leading-order approximation of the quantum circuit depth from existing top-down-based algorithms [10, 40] for sub-states with $s$ qubits. The exact expression depends on which of the two algorithms is used. The summation of the second term (stage 2) is the divide-and-conquer circuit depth from split level $s + 1$ to $n$. Similarly, the first term in Eq. (12) is the number of qubits occupied by all first stage sub-states and the second term is the number of qubits used by the second stage.

There are three noteworthy configuration values for the parameter $s$ (see Table 1). Setting $s = \log_2(\sqrt{N})$ achieves asymptotic sublinearity, and $s = 1$ or $s = \log_2(N)$ recovers bottom-up or top-down approaches.

The condition for quadratic reduction in both depth and width is obtained through asymptotic analysis of the minimum distance between Eqs. (11) and (12). The first (second) equation is a monotonically increasing (decreasing) function $\forall s \in \{x \in \mathbb{R} | 4 \leq x \leq \log_2(N)\}$ and there is a point $s = n/2$ where the distance is zero when $N \to \infty$, which leads to Theorem 2.

**Theorem 2** Algorithm 5 with $s = n/2$ generates a quantum circuit with sublinear depth $O_d(\sqrt{N})$ and width $O_w(\log_2(N)\sqrt{N})$. 

\[ Springer \]
When \( N \leq 8 \) a top-down approach (\( s = \log_2(N) \)) should be used, since space and depth both decrease as \( s \) increases in the interval \( s \in [1..3] \) (see Table 3 for a numerical example). Circuit depth increases only when \( s \geq 4 \).

### 3.2 Experiment

To evaluate the bidirectional method, proof-of-principle experiments were performed on a classical simulator provided by IBM, four superconducting-qubit-based quantum devices provided by IBM, and an ion-trap based quantum device provided by IonQ. These are named as ibmq_qasm_simulator, ibmq_rome, ibmq_santiago, ibmq_casablanca, and ibmq_jakarta, and IonQ, respectively. The experiments aim to load the following 8 and 16-dimensional real input vectors:

\[
(\sqrt{0.03}, \sqrt{0.06}, \sqrt{0.15}, \sqrt{0.05}, \sqrt{0.1}, \sqrt{0.3}, \sqrt{0.2}, \sqrt{0.11})
\]

and

\[
(\sqrt{0.01}, \sqrt{0.02}, \sqrt{0.04}, \sqrt{0.02}, \sqrt{0.07}, \sqrt{0.08}, \sqrt{0.04}, \sqrt{0.01}, \\
\sqrt{0.08}, \sqrt{0.02}, \sqrt{0.21}, \sqrt{0.09}, \sqrt{0.12}, \sqrt{0.08}, \sqrt{0.05}, \sqrt{0.06}).
\]

Three configurations of the bidirectional method are compared, namely top-down (\( s = n \)), bottom-up (\( s = 1 \)), and sublinear (\( s = \lceil n/2 \rceil \)). The first case uses the least number of qubits \( O_w(\log_2(N)) \) and maximum depth \( O_d(N) \). In the second configuration, depth is minimum \( O_d(\log_2^2(N)) \) and the number of qubits is maximum \( O_u(N) \). The last configuration uses the best trade-off between the quantum circuit depth and width and achieves the sublinear scaling for both. In this case, the quantum circuit depth and width both grow sublinearly.

Note that Algorithms 3 and 4 allocate logical qubits as they are needed, without concerning their assignment to physical qubits of the quantum device. For NISQ devices with limited quantum device coupling map, the logical to physical qubit mapping should be optimized in order to minimize the overhead in the quantum circuit depth and the number of gates. The Python code used in this work for implementing Algorithm 3 employs functions \( \text{ucry} \) and \( \text{ucrz} \) from Qiskit. These functions are called uniformly controlled rotations (or multiplexers), and the code is based on the work of Shende et al. [10].

Table 2 lists the experimental results, presenting the number of runs per device and dimensionality of the input vector. The ibmq_rome and ibmq_santiago devices have only five qubits, and due to this limitation they are not suitable to encode the eight-dimensional vector with the bottom-up configuration (i.e., \( s = 1 \)) or to perform sublinear (i.e., \( s = \lceil n/2 \rceil \)) and bottom-up experiments to encode the 16-dimensional vector. None of the quantum devices used in this work has the capacity to run the bottom-up configuration for the 16-dimensional input vector, which requires at least 15 qubits (i.e., \( N - 1 \) qubits).
Table 2 Results of the BDSP experiments that encode $N$-dimensional input vectors in the amplitudes of quantum states using a classical simulator and quantum devices for $N = \{8, 16\}$

| Device                        | $N$ | $s$ | Runs | MAE   |
|-------------------------------|-----|-----|------|-------|
| ibmq_qasm_simulator 32 qubits | 8   | 1   | 5    | 0.0016|
|                               | 2   | 5   |      | 0.0005|
|                               | 3   | 5   |      | 0.0015|
|                               | 4   | 5   |      | 0.0010|
| ibmq_rome 5 qubits            | 8   | 2   | 10   | 0.0577|
|                               | 3   | 10  |      | 0.0429|
|                               | 4   | 5   |      | 0.0409|
| ibmq_santiago 5 qubits        | 8   | 2   | 10   | 0.0464|
|                               | 3   | 10  |      | 0.0233|
|                               | 16  | 4   | 10   | 0.0225|
| ibmq_casablanca 7 qubits      | 8   | 1   | 10   | 0.0710|
|                               | 2   | 10  |      | 0.0691|
|                               | 3   | 10  |      | 0.0213|
| ibmq_jakarta 7 qubits         | 8   | 1   | 10   | 0.0594|
|                               | 2   | 10  |      | 0.0497|
|                               | 3   | 10  |      | 0.0289|
| IonQ 11 qubits                | 8   | 1   | 5    | 0.0455|
|                               | 2   | 5   |      | 0.0242|
|                               | 3   | 5   |      | 0.0217|
|                               | 16  | 2   | 5    | 0.0261|
|                               | 4   | 5   |      | 0.0107|

The acronym MAE stands for mean absolute error. The bold font indicates the smallest MAE, and hence, the best performance, among different configurations of $s$ for each device and input vector.

Figure 5 presents the average output of the experiments with 8 and 16-dimensional input vectors. The height of blue and red bars is an average value obtained from a number of repetitions shown in the runs column in Table 2, and the error bars represent the standard deviation. The height of the yellow bar is the experimental result averaged over all quantum devices.

Table 3 and Fig. 6 show the trade-off between quantum circuit depth, width and the number of CNOT gates as $s$ is varied for randomly generated target vectors of various sizes. As expected through the analysis of the number of CNOT gates and the circuit depth in Table 3, the experimental results in Table 2 and Fig. 5 show performance favoring the top-down configuration ($s = n$) for small input sizes ($N < 64$) due to the smaller number of CNOT gates and the smaller or approximately equal depth of the circuit. The number of CNOT gates, circuit depth, and number of qubits all decrease as $s$ progresses to $s = 3$. The depth starts to increase when $s > 3$, as previously implied by Eq. (11). The comparison employs the mean absolute error (MAE). For each device and input size, the ranking is established such that a smaller MAE indicates better performance (see Table 2).
Fig. 5 Experimental results with 8- and 16-dimensional input vectors. Blue and red bars indicate, respectively, the ideal results and the ibm_qasm_simulator results. Yellow bars indicate the output average values from the experiments on all quantum devices. Error bars are the standard deviation (Color figure online)

Data in Table 3 and Fig. 6 were obtained using the *transpile* method from the Quantum Information Science Kit (Qiskit [51]) version 0.26.2 to decompose the circuits into physical single-qubit gates and the CNOT gate. These circuits were generated by the bidirectional algorithm with random complex input vectors.

### 4 Conclusion

Many quantum algorithms require an efficient way to load high-dimensional classical data to a quantum state. Existing state preparation methods, such as top-down and bottom-up approaches, require at least one quantum circuit resource between depth and width to grow linearly with the problem size. The BDSP algorithm presented in this work provides a general framework for configuring the trade-off between these resources that can be useful to manage them on NISQ devices. Looking at the state preparation algorithms as a walk on the state tree (see Sect. 2.1), the BDSP algorithm constitutes a systematic way to walk in two opposite directions. Previous methods are based on walking only in one direction. The bidirectional algorithm comes with a free parameter $s \in [1, n]$ that determines the balance between the top-down and the bottom-up approaches. At two extreme cases of setting $s = n$ and $s = 1$, the top-down and the bottom-up approaches are, respectively, recovered. At the equilibrium point $s = \lceil n/2 \rceil$, quadratic reduction in both quantum circuit depth and width can be achieved. The configuration parameter can be viewed as a hyperparameter that can tune circuit sizes and the number of CNOT gates according to the compound of application and hardware properties. The BDSP method is validated and demonstrated through experiments performed on five real quantum devices. The experiments behaved as expected, according to the asymptotic and numerical analyses of the circuit complexity.
### Table 3  Exchange between circuit depth, width (qubits), and number of CNOTs by adjusting the parameter $s$ (split)

| $s$ | $N = 8$ |         |         | $N = 16$ |         |         | $N = 32$ |         | $N = 64$ |         |
|-----|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
|     | CNOTs   | Depth   | Qubits  | CNOTs   | Depth   | Qubits  | CNOTs   | Depth   | Qubits  | Depth   |
| 1   | 28      | 31      | 7       | 77      | 58      | 15      | 182     | 93      | 31      | 136     |
| 2   | 18      | 24      | 5       | 57      | 51      | 11      | 142     | 86      | 23      | 129     |
| 3   | 10      | 20      | 3       | 41      | 48      | 7       | 110     | 83      | 15      | 126     |
| 4   | 26      | 51      | 4       | 80      | 87      | 9       | 195     | 130     | 19      | 158     |
| 5   | 58      | 114     | 5       | 151     | 158     | 11      |         |         |         |         |
| 6   | 122     | 241     | 6       |         |         |         |         |         |         |         |

$s$ can be interpreted as a hyperparameter to fine-tune the encoding circuit to hardware characteristics such as relaxation time, dephasing time, and the CNOT gate error. Circuit transpilation uses Qiskit’s transpile function with the parameter `basis_gates = ['u1', 'u2', 'u3', 'cx']` and does not target a backend device.
Exchange between circuit depth, width (number of qubits), and number of CNOTs to load a $2^n$-dimensional complex vector into a quantum computer by adjusting parameter $s$. The increasing number of CNOTs at lower depths is a consequence of exchanging computational time for space, given the combination of distant states. It also indicates an increase in concurrent operations.

A possible future work is to investigate whether the quantum circuit cost of the DCSP part can be further reduced. Note that the structure of CSWAP operations in the DCSP step only depends on the dimensionality of the dataset $N$. Hence, the CSWAP operations can be interpreted as a single layer of fixed operation. Decomposing this fixed operation more efficiently than the naive application of CSWAP gates would achieve further reduction in the quantum circuit depth. Finally, developing a BDSP method whose complexity depends on the number of nonzero elements instead of the size of the entire data vector also remains as an important future work. Such a method will provide a significant improvement for sparse data.

Acknowledgements This work is based upon research supported by CNPq (Grant Nos. 308730/2018-6, 306727/2017-0, 409415/2018-9 and 421849/2016-9), CAPES – Finance Code 001, FACEPE (Grant No. IBPG-0834–1.03/19), National Research Foundation of Korea (Grant Nos. 2019M3E4A1079666, 2021M3H3A1038085 and 2022M3E4A1074591), the South African Research Chair Initiative, Grant No. 64812, of the Department of Science and Innovation and the National Research Foundation (NRF). Support of the NICIS (National Integrated Cyberinfrastructure System) e-research grant QICSA is kindly acknowledged. We acknowledge the use of IBM Quantum services for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum team.
Data availability statement  The sites https://github.com/qclib/qclib-papers and https://github.com/qclib/qclib contain all the data and the software generated during the current study.

Declarations

Conflict of interest  The authors declare no competing interests.

A Numerical example

The following case study illustrates how the BDSP algorithm (Algorithm 5) encodes the example input vector \( \mathbf{x} = (0, \sqrt{2/8} e^{-i\pi/7}, \sqrt{3/8} e^{-i\pi/3}, 0, 0, 0, \sqrt{3/8}, 0) \). Initially (step 1 of Algorithm 5), Algorithm 1 is employed, as detailed below, to produce the state decomposition tree (Fig. 7b).

Iteration #1 \( (n = 3; k = 1; i = 1 \ldots 4): \)

\[
\begin{align*}
\eta_{1,1} &= \sqrt{|x_0|^2 + |x_1|^2} = \sqrt{2/8} \\
\eta_{2,1} &= \sqrt{|x_2|^2 + |x_3|^2} = \sqrt{3/8} \\
\eta_{3,1} &= \sqrt{|x_4|^2 + |x_5|^2} = 0 \\
\eta_{4,1} &= \sqrt{|x_6|^2 + |x_7|^2} = \sqrt{3/8}
\end{align*}
\]

\[
\begin{align*}
\Omega_{1,1} &= (\omega_0 + \omega_1) = -\pi/7 \\
\Omega_{2,1} &= (\omega_2 + \omega_3) = -\pi/3 \\
\Omega_{3,1} &= (\omega_4 + \omega_5) = 0 \\
\Omega_{4,1} &= (\omega_6 + \omega_7) = 0
\end{align*}
\]
Iteration \#2 \((n = 3; k = 2; i = 1 \ldots 2)\):

\[
\begin{align*}
\eta_{1,2} &= \sqrt{|x_0|^2 + |x_1|^2 + |x_2|^2 + |x_3|^2} = \sqrt{\eta_{1,1}^2 + \eta_{2,1}^2} = \sqrt{5/8} \\
\eta_{2,2} &= \sqrt{|x_4|^2 + |x_5|^2 + |x_6|^2 + |x_7|^2} = \sqrt{\eta_{3,1}^2 + \eta_{4,1}^2} = \sqrt{3/8} \\
\Omega_{1,2} &= (\omega_0 + \omega_1 + \omega_2 + \omega_3)/2 = (\Omega_{1,1} + \Omega_{2,1})/2 = -5\pi/21 \\
\Omega_{2,2} &= (\omega_4 + \omega_5 + \omega_6 + \omega_7)/2 = (\Omega_{3,1} + \Omega_{4,1})/2 = 0
\end{align*}
\]

Iteration \#3 \((n = 3; k = 3; i = 1)\):

\[
\begin{align*}
\eta_{1,3} &= \sqrt{|x_0|^2 + |x_1|^2 + |x_2|^2 + |x_3|^2 + |x_4|^2 + |x_5|^2 + |x_6|^2 + |x_7|^2} \\
&= \sqrt{\eta_{1,2}^2 + \eta_{2,2}^2} = 1 \\
\Omega_{1,3} &= (\omega_0 + \omega_1 + \omega_2 + \omega_3 + \omega_4 + \omega_5 + \omega_6 + \omega_7)/4 \\
&= (\Omega_{1,2} + \Omega_{2,2})/2 = -5\pi/42
\end{align*}
\]

Then (step 2 of Algorithm 5), Algorithm 2 uses the state tree data to generate the angle tree (Fig. 7b). The iterations of the procedure are detailed below.

Iteration \#1 \((n = 3; v = 3; j = 1)\):

\[
\begin{align*}
\beta_{1,3} &= \eta_{2,2}/\eta_{1,3} = \sqrt{3/8} \\
\lambda_{1,3} &= \Omega_{2,2} - \Omega_{1,3} = 5\pi/42
\end{align*}
\]

Iteration \#2 \((n = 3; v = 2; j = 1 \ldots 2)\):

\[
\begin{align*}
\beta_{1,2} &= \eta_{2,1}/\eta_{1,2} = \sqrt{3/5} \\
\beta_{2,2} &= \eta_{4,1}/\eta_{2,2} = 1 \\
\lambda_{1,2} &= \Omega_{2,1} - \Omega_{1,2} = -2\pi/21 \\
\lambda_{2,2} &= \Omega_{4,1} - \Omega_{2,2} = 0
\end{align*}
\]

Iteration \#3 \((n = 3; v = 1; j = 1 \ldots 4, \eta_{i,0} = |x_{i-1}|, \Omega_{i,0} = 2\omega_{i-1})\):

\[
\begin{align*}
\beta_{1,1} &= \eta_{2,0}/\eta_{1,1} = 1 \\
\beta_{2,1} &= \eta_{4,0}/\eta_{2,1} = 0 \\
\beta_{3,1} &= \eta_{6,0}/\eta_{3,1} = 0 \\
\beta_{4,1} &= \eta_{8,0}/\eta_{4,1} = 0 \\
\lambda_{1,1} &= \Omega_{2,0} - \Omega_{1,1} = -\pi/7 \\
\lambda_{2,1} &= \Omega_{4,0} - \Omega_{2,1} = \pi/3 \\
\lambda_{3,1} &= \Omega_{6,0} - \Omega_{3,1} = 0 \\
\lambda_{4,1} &= \Omega_{8,0} - \Omega_{4,1} = 0
\end{align*}
\]
Setting the parameter $s = 2$ separates the bottom part of the angle tree into two sub-trees of height 2, and step 3 of Algorithm 5 creates a quantum circuit with $n = (s + 1)2^{n-s} - 1 = 5$ qubits. Step 4 (named Stage I) prepares $2^{n-s} = 2$ sub-states, $|\psi\rangle_{1,3}$ and $|\phi\rangle_{2,5}$, with 2 qubits each (the sub-state indexes indicate the circuit’s corresponding qubits) using amplitude encoding (see Eqs. (5) and (6) from Sect. 2.2 and Refs. [29, 40, 53, 54] for details on amplitude encoding) with sub-trees as input (Fig. 8).

Preparing sub-state $|\psi\rangle_{1,3}$:

$$|\psi\rangle = |0\rangle_0|0\rangle_1|0\rangle_3|0\rangle_2|1\rangle_5|0\rangle_4|0\rangle_6|0\rangle_7$$

$$|\psi\rangle = |0\rangle_0|0\rangle_1|0\rangle_3|0\rangle_2|1\rangle_5|0\rangle_4|0\rangle_6|0\rangle_7$$

$$|\psi\rangle_{1,3} = e^{-i\frac{\lambda_2}{2}}\sqrt{1 - |\beta_{1,2}|^2}|0\rangle_1 + e^{i\frac{\lambda_2}{2}}|\beta_{1,2}|1\rangle_1$$

Preparing sub-state $|\phi\rangle_{2,5}$:

$$|\phi\rangle = e^{-i\frac{\lambda_2}{2}}\sqrt{1 - |\beta_{2,2}|^2}|0\rangle_2 + e^{i\frac{\lambda_2}{2}}|\beta_{2,2}|1\rangle_2$$

$$|\phi\rangle = e^{-i\frac{\lambda_2}{2}}\sqrt{1 - |\beta_{2,2}|^2}|0\rangle_2 + e^{i\frac{\lambda_2}{2}}|\beta_{2,2}|1\rangle_2$$

$$|\phi\rangle_{2,5} = e^{-i\frac{\lambda_3}{2}}\sqrt{1 - |\beta_{3,1}|^2}|0\rangle_5 + e^{i\frac{\lambda_3}{2}}|\beta_{3,1}|1\rangle_5$$

$$|\phi\rangle_{2,5} = e^{-i\frac{\lambda_3}{2}}\sqrt{1 - |\beta_{3,1}|^2}|0\rangle_5 + e^{i\frac{\lambda_3}{2}}|\beta_{3,1}|1\rangle_5$$

Fig. 8 Circuit generated by the top-down strategy (Algorithm 3) to prepare two 2-qubit sub-states for the example input vector $x$. 

Preparing sub-state $|\psi\rangle_{1,3}$:

$$|\psi\rangle_{1,3} = e^{-i\frac{\lambda_2}{2}}\sqrt{1 - |\beta_{1,2}|^2}|0\rangle_1 + e^{i\frac{\lambda_2}{2}}|\beta_{1,2}|1\rangle_1$$

Preparing sub-state $|\phi\rangle_{2,5}$:

$$|\phi\rangle_{2,5} = e^{-i\frac{\lambda_3}{2}}\sqrt{1 - |\beta_{3,1}|^2}|0\rangle_5 + e^{i\frac{\lambda_3}{2}}|\beta_{3,1}|1\rangle_5$$

$$|\phi\rangle_{2,5} = e^{-i\frac{\lambda_3}{2}}\sqrt{1 - |\beta_{3,1}|^2}|0\rangle_5 + e^{i\frac{\lambda_3}{2}}|\beta_{3,1}|1\rangle_5$$
Fig. 9 Circuit generated by the bottom-up strategy (Algorithm 4) to combine two 2-qubit sub-states for the example input vector \( x \)

\[
|\psi\rangle_{1,3} = \frac{1}{2} \left( |0\rangle_{0} R_y(1.32) R_z(5\pi/42) |\phi\rangle_{2,5} + |1\rangle_{0} |\phi\rangle_{2,5} \right)
\]

\[
|\phi\rangle_{2,5} = |0\rangle_{2,5}
\]

Step 5 of Algorithm 5 (named Stage 2) uses the bottom-up approach (see Eq. (8) from Sect. 2.3 and Refs. [43, 53] for details on divide-and-conquer quantum state preparation) to combine the two states (Eqs. (A1) and (A2)) prepared in Step 4 (Stage 1) into the final state \(|\Psi\rangle\) using the remaining qubit (Fig. 9).

Preparing final state \(|\Psi\rangle\):

\[
|\Psi\rangle = e^{-i \frac{\lambda_1}{2}} \sqrt{1 - |\beta_{1,3}|^2} |\psi\rangle_{1,3} |\phi\rangle_{2,5} + e^{i \frac{\lambda_1}{2}} |\beta_{1,3}| |\phi\rangle_{1,3} |\psi\rangle_{2,5}
\]

\[
= e^{-i \frac{5\pi}{24}} \left( e^{-i \frac{\pi}{7}} \sqrt{\frac{2}{8}} |001\rangle_{0,1,3} + e^{-i \frac{\pi}{7}} \sqrt{\frac{3}{8}} |010\rangle_{0,1,3} \right) |\phi\rangle_{2,5}
\]

\[
+ e^{i \frac{5\pi}{24}} \left( e^{i \frac{\pi}{7}} \sqrt{\frac{3}{8}} |110\rangle_{0,1,3} \right) |\psi\rangle_{2,5}
\]

\[
= e^{-i \frac{\pi}{7}} \sqrt{\frac{2}{8}} |001\rangle |\phi\rangle + e^{-i \frac{\pi}{7}} \sqrt{\frac{3}{8}} |010\rangle |\phi\rangle + \sqrt{\frac{3}{8}} |110\rangle |\psi\rangle
\]

The global phase \( e^{i \frac{5\pi}{24}} \) in Eq. (A3) can be easily calculated \( e^{-i \sum_{j=0}^{N-1} \alpha_j / N} \) and eliminated, as prescribed in Ref. [40]. The indexes have been removed from Eq. (A3) last expression to improve readability.

Figure 10 presents the complete circuit constructed by Algorithm 5 using the angle tree split at level \( s = 2 \), operating only multi-controlled rotations and CSWAP gates.
Fig. 10 Complete circuit generated by the bidirectional strategy (Algorithm 5) for the example input vector \( x \)

### B Pseudocode

Pseudocodes 1 to 5 express algorithms 1 to 5. Pseudocodes 1 and 2 construct the tree representations of the state preparation algorithms, namely the state tree and the angle tree, as described in Sect. 2.1. Pseudocodes 3 and 4, which algorithms are explained in Sects. 2.2 and 2.3, build quantum circuits using top-down and bottom-up approaches for encoding a complex input vector into the amplitudes of a quantum state. Pseudocode 5 employs pseudocodes 1 to 4 and expresses the bidirectional state preparation algorithm (Sect. 3, Algorithm 5).

Lines 5 and 6 of Pseudocode 5 indicate the two stages of the BDSP algorithm. Line 5 at function top_down_tree_walk performs the first stage preparing the sub-states expected by the next stage, equivalent to what would be generated by bottom-up DCSP up to the tree split, but with the absence of ancilla due to the top-down approach. Line 6 at function bottom_up_tree_walk performs the second stage, starting at level \( s + 1 \) with the sub-states initialized by the previous stage. Line 3 at function top_down_tree_walk configures the recurrence so that at split level \( s \) it divides the angle tree into \( 2^{n-s} \) (number of nodes at split level \( s \)) sub-trees of height \( s \), loading all these sub-trees concurrently using the top-down strategy. Lines 11 and 12 of function bottom_up_tree_walk initialize \( 2^{n-s} - 1 \) qubits exclusive to the second stage with values \( R_y(\alpha_j,v) \) and \( R_z(\lambda_j,v) \). Then, function cs swaps combine the states through CSWAP gates controlled by the nodes above level \( s \). With the tree described in Fig. 4a and \( s = 2 \), the bidirectional procedure (Pseudocode 5) generates the circuit presented in Fig. 4c.
Pseudocode 1: Generate a state tree by the decomposition of an amplitude input
vector

1 state_decomposition(nqubits, data):
   input: Number of qubits (nqubits) required to generate a state with the same length as the data
   vector (2^nqubits).
   input: A list (data) representing the state to be decomposed, with exactly 2^nqubits pairs (index,
amplitude).
   output: Root of the state tree.
   // Initialize an auxiliary vector new_nodes with data vector amplitudes
   new_nodes = []
   for k ← 0 to length(data) − 1 do
      node.index = data[k].index
      node.level = nqubits
      node.amplitude = data[k].amplitude
      new_nodes[k] = node
   // Build the state tree
   for level ← nqubits to 1 step −1 do
      nodes = new_nodes
      new_nodes = []
      for k ← 0 to length(nodes) − 1 step 2 do
         mag = √|nodes[k].amplitude|^2 + |nodes[k+1].amplitude|^2
         arg = (∠nodes[k].amplitude + ∠nodes[k+1].amplitude)/2
         node.index = nodes[k].index // 2
         node.level = level
         node.mag = mag
         node.arg = arg
         node.left = nodes[k]
         node.right = nodes[k+1]
         new_nodes[k//2] = node
   return new_nodes[0]; // return tree root
**Pseudocode 2:** Generate a angle tree that will be used to perform the state preparation

```
angle_tree(state_tree):
    input: An output of state_decomposition function (state_tree).
    output: Tree with angles that will be used to perform the state preparation.

angle_y, angle_z = 0
if state_tree.right ≠ null then
    mag = 0
    if state_tree.mag ≠ 0 then
        mag = state_tree.right.mag / state_tree.mag
    arg = state_tree.right.arg - state_tree.arg
    angle_y = 2 arcsin(mag)
    angle_z = 2 arg
    node.index = state_tree.index
    node.level = state_tree.level
    node.angle_y = angle_y
    node.angle_z = angle_z
if state_tree.left ≠ null & !is_leaf(state_tree.left) then
    node.left = angle_tree(state_tree.left)
if state_tree.right ≠ null & !is_leaf(state_tree.right) then
    node.right = angle_tree(state_tree.right)
return node
```
**Pseudocode 3:** Construct a circuit that perform a top-down state preparation for the input vector \( state \). The intended quantum state is encoded on the output qubits.

```plaintext
1 top_down_tree_walk(angle_tree, circuit, start_level, control_nodes=null, target_nodes=null):
   input : An output of angle_tree function (angle_tree).
   input : A quantum circuit to apply the top-down encoding (circuit).
   input : The tree level to start the walk (start_level).
   input : Used in the recursive calls (control_nodes).
   input : Used in the recursive calls (target_nodes).
   output: circuit after the application of the top-down encoding.
2 if angle_tree ≠ null then
3   if angle_tree.level < start_level then
4     top_down_tree_walk(angle_tree.left, circuit, start_level)
5     top_down_tree_walk(angle_tree.right, circuit, start_level)
6   else
7     angle_tree.qubit = add_qubit(circuit)
8     if target_nodes == null then
9       control_nodes = [] ;  // initialize the controls list
10      target_nodes[0] = angle_tree ;  // start by the sub-tree root
11     uniformly_controlled_rotation(circuit, control_nodes, target_nodes)
12     append(control_nodes, angle_tree) ;  // add curr. node to the
13     target_nodes = children(target_nodes) ;  // all the nodes in the next
14     if angle_tree.left ≠ null then
15       top_down_tree_walk(angle_tree.left, circuit, start_level, control_nodes,
16      target_nodes)
17     else
18       top_down_tree_walk(angle_tree.left, circuit, start_level, control_nodes,
19      target_nodes)
20 initialize_top_down(circuit, state):
21   nqubits = log_2(length(state))
22   state_tree = state_decomposition(nqubits, state)
23   angle_tree = angle_tree (state_tree)
24   top_down_tree_walk(angle_tree, circuit, 0)
25   output_nodes = left_view(angle_tree)
26   for k ← 0 to nqubits − 1 do
27     output_qubits[k] = output_nodes[k].qubit
28   return output_qubits
```

Pseudocode 4: Construct a circuit that perform a bottom-up state preparation for the input vector \textit{state}. The intended quantum state is encoded on the \textit{output_qubits}.

```python
cswaps(angle_tree, circuit):
    input : An output of angle_tree function (angle_tree).
    input : A quantum circuit to apply the cswaps (circuit).
    output: circuit after the application of the cswaps.
    left = angle_tree.left
    right = angle_tree.right
    while left \neq null & right \neq null do
        circuit.cswap(angle_tree.qubit, left.qubit, right.qubit)
        left = left.left
        right = right.left

bottom_up_tree_walk(state_tree, circuit, start_level):
    input : An output of state_decomposition function (state_tree).
    input : A quantum circuit to apply the bottom-up encoding (circuit).
    input : The tree level to start the bottom-up walk (start_level).
    output: circuit after the application of the bottom-up encoding.
    if angle_tree \neq null & angle_tree.level < start_level then
        angle_tree.qubit = add_qubit(circuit)
        circuit.ry(angle_tree.angle_y, angle_tree.qubit)
        circuit.rz(angle_tree.angle_z, angle_tree.qubit)
        bottom_up_tree_walk(angle_tree.left, circuit, start_level)
        bottom_up_tree_walk(angle_tree.right, circuit, start_level)
        cswaps(angle_tree, circuit)

initialize_bottom_up(circuit, state):
    nqubits = \log_2(\text{length(state)})
    state_tree = state_decomposition(nqubits, state)
    angle_tree = angle_tree(state_tree)
    bottom_up_tree_walk(angle_tree, circuit, nqubits)
    output_nodes = left_view(angle_tree)
    for k \leftarrow 0 to nqubits - 1 do
        output_qubits[k] = output_nodes[k].qubit
    return output_qubits
```

Springer
Pseudocode 5: Construct a circuit that perform a bidirectional state preparation for the input vector \textit{state}. The intended quantum state is encoded on the output qubits.

```python
initialize_bidirectional(circuit, state, split):
    nqubits = \log_2(\text{length}(state))
    state_tree = state_decomposition(nqubits, state)
    angle_tree = \text{angle_tree}(state_tree)
    top_down_tree_walk(angle_tree, circuit, nqubits - split);  // stage 1
    bottom_up_tree_walk(angle_tree, circuit, nqubits - split);  // stage 2
    output_nodes = left_view(angle_tree)
    for k ← 0 to nqubits − 1 do
        output_qubits[k] = output_nodes[k].qubit
    return output_qubits
```

References

1. Feynman, R.P.: Simulating physics with computers. Int. J. Theor. Phys. 21(6–7), 467–488 (1982)
2. Deutsch, D., Jozsa, R.: Rapid solution of problems by quantum computation. Proc. R. Soc. Lond. Ser. A Math. Phys. Sci. 439(1907), 553–558 (1992)
3. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing. STOC ’96, pp. 212–219. Association for Computing Machinery, Philadelphia (1996)
4. Simon, D.R.: On the power of quantum computation. SIAM J. Comput. 26(5), 1474–1483 (1997)
5. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. 41(2), 303–332 (1999)
6. Aaronson, S.: Read the fine print. Nat. Phys. 11(4), 291–293 (2015)
7. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature 549(7671), 195–202 (2017)
8. Leymann, F., Barzen, J.: The bitter truth about gate-based quantum algorithms in the NISQ era. Quantum Sci. Technol. 5(4), 044007 (2020)
9. Tang, E.: Quantum principal component analysis only achieves an exponential speedup because of its state preparation assumptions. Phys. Rev. Lett. 127, 060503 (2021)
10. Shende, V.V., Bullock, S.S., Markov, I.L.: Synthesis of quantum-logic circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 25(6), 1000–1010 (2006)
11. Hogg, T., Huberman, B.A., Williams, C.P.: Phase transitions and the search problem. Artif. Intell. 81(1), 1–15 (1996) (Frontiers in problem solving: phase transitions and complexity)
12. Terhal, B.M., Smolin, J.A.: Single quantum querying of a database. Phys. Rev. A 58(3), 1822–1826 (1998)
13. Zidan, M., Eleuch, H., Abdel-Aty, M.: Non-classical computing problems: toward novel type of quantum computing problems. Results Phys. 21, 103536 (2021)
14. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. 103, 150502 (2009)
15. Grover, L.K.: Synthesis of quantum superpositions by quantum computation. Phys. Rev. Lett. 85(6), 1334–1337 (2000)
16. Sanders, Y.R., Low, G.H., Scherer, A., Berry, D.W.: Black-box quantum state preparation without arithmetic. Phys. Rev. Lett. 122, 020502 (2019)
17. Trugenberger, C.A.: Probabilistic quantum memories. Phys. Rev. Lett. 87(6), 067901 (2001)
18. Ventura, D., Martinez, T.: Quantum associative memory. Inf. Sci. 124(1), 273–296 (2000)
20. Trugenberger, C.A.: Quantum pattern recognition. Quantum Inf. Process. 1(6), 471–493 (2002)
21. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Phys. Rev. Lett. 100, 160501 (2008)
22. Silva, A., de Oliveira, W., Ludermir, T.: A weightless neural node based on a probabilistic quantum memory. In: 2010 Eleventh Brazilian Symposium on Neural Networks, pp. 259–264. IEEE, Sao Paulo (2010). ISSN: 2375-0235
23. de Paula Neto, F.M., da Silva, A.J., de Oliveira, W.R., Ludermir, T.B.: Quantum probabilistic associative memory architecture. Neurocomputing 351, 101–110 (2019)
24. Park, D.K., Petruccione, F., Rhee, J.-K.K.: Circuit-based quantum random access memory for classical data. Sci. Rep. 9(1), 3949 (2019)
25. Zidan, M., Abdel-Aty, A.-H., Khalil, A., Abdel-Aty, M., Eleuch, H.: A novel efficient quantum random access memory. IEEE Access 9, 151775–151780 (2021)
26. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum algorithms for supervised and unsupervised machine learning (2013). arXiv:1307.0411 [quant-ph]
27. Stoudenmire, E., Schwab, D.J.: Supervised learning with tensor networks. In: Lee, D., Sugiyama, M., Luxburg, U.-.G., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29, p. 9. Curran Associates, Inc., Centre Convencions Internacional Barcelona, Barcelona (2016)
28. Schuld, M., Fingerhuth, M., Petruccione, F.: Implementing a distance-based classifier with a quantum interference circuit. EPL (Europhys. Lett.) 119(6), 60002 (2017)
29. Scott, M., Petruccione, F.: Supervised Learning with Quantum Computers, 1st ed. 2018 edn. Quantum Science and Technology. Springer, Cham (2018)
30. Benedetti, M., Lloyd, E., Sack, S., Fiorentini, M.: Parameterized quantum circuits as machine learning models. Quantum Sci. Technol. 4, 043001 (2019)
31. Levine, Y., Sharir, O., Cohen, N., Shashua, A.: Quantum entanglement in deep learning architectures. Phys. Rev. Lett. 122(6), 065301 (2019)
32. Blank, C., Park, D.K., Rhee, J.-K.K., Petruccione, F.: Quantum classifier with tailored quantum kernel. npj Quantum Inf. 6(1), 1–7 (2020)
33. Lloyd, S., Mohseni, M., Rebentrost, P.: Quantum principal component analysis. Nat. Phys. 10(9), 631–633 (2014)
34. Childs, A.M., Kothari, R., Somma, R.D.: Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. SIAM J. Comput. 46(6), 1920–1950 (2017)
35. Wossnig, L., Zhao, Z., Prakash, A.: Quantum linear system algorithm for dense matrices. Phys. Rev. Lett. 120, 050502 (2018)
36. Recht, P., Recht, P., Stothers, A., Marvian, I., Lloyd, S.: Quantum singular-value decomposition of nonsparse low-rank matrices. Phys. Rev. A 97(1), 012327 (2018)
37. Mitchell, T.M.: Machine Learning. McGraw-Hill Series in Computer Science, New York (2013)
38. Bartlett, D., Martinez, T.: Initializing the amplitude distribution of a quantum state. Found. Phys. Lett. 12(6), 547–559 (1999)
39. Long, G.-L., Sun, Y.: Efficient scheme for initializing a quantum register with an arbitrary superposed state. Phys. Rev. A 64(1), 014303 (2001)
40. Mottnen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Transformation of quantum states using uniformly controlled rotations. Quantum Inf. Comput. 5(6), 467–473 (2005)
41. Plesch, M., Brukner, C.: Quantum-state preparation with universal gate decompositions. Phys. Rev. A 83(3), 032302 (2011)
42. Cortese, J.A., Braje, T.M.: Loading classical data into a quantum computer (2018)
43. Araujo, I.F., Park, D.K., Petruccione, F., da Silva, A.J.: A divide-and-conquer algorithm for quantum state preparation. Sci. Rep. 11(1), 6329 (2021)
44. Park, D.K., Sinayskiy, I., Fingerhuth, M., Petruccione, F., Rhee, J.-K.K.: Parallel quantum trajectories via forking for sampling without redundancy. New J. Phys. 21(8), 083024 (2019)
45. Low, G.H., Kliuchnikov, V., Schaeffer, L.: Trading T-gates for dirty qubits in state preparation and unitary synthesis (2018)
46. Zoufal, C., Lucchi, A., Woerner, S.: Quantum generative adversarial networks for learning and loading random distributions. npj Quantum Inf. 5(1), 1–9 (2019)
47. Kuzmin, V.V., Silvi, P.: Variational quantum state preparation via quantum data bases. Quantum 4, 290 (2020)
48. IBM: IBM’s roadmap for scaling quantum technology (2020). https://research.ibm.com/blog/ibm-quantum-roadmap
49. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum 2, 79 (2018)
50. Xin, T., Wei, S., Cui, J., Xiao, J., Arrazola, I.N., Lamata, L., Kong, X., Lu, D., Solano, E., Long, G.: Quantum algorithm for solving linear differential equations: theory and experiment. Phys. Rev. A 101, 032307 (2020)
51. Aleksandrowicz, G., et al.: Qiskit: an open-source framework for quantum computing (2021)
52. Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M.S., Ahmed, S., Arrazola, J.M., Blank, C., Delgado, A., Jahangiri, S., McKiernan, K., Meyer, J.J., Niu, Z., Száva, A., Killoran, N.: PennyLane: automatic differentiation of hybrid quantum-classical computations (2020)
53. Schuld, M., Petruccione, F.: Machine Learning with Quantum Computers. Quantum Science and Technology. Springer, Cham (2021)
54. Bergholm, V., Vartiainen, J.J., Möttönen, M., Salomaa, M.M.: Quantum circuits with uniformly controlled one-qubit gates. Phys. Rev. A 71(5), 052330 (2005)

**Publisher’s Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.

**Authors and Affiliations**

Israel F. Araujo\(^1,2,3\) · Daniel K. Park\(^2,3\) · Teresa B. Ludermir\(^1\) · Wilson R. Oliveira\(^4\) · Francesco Petruccione\(^5,6,7\) · Adenilton J. da Silva\(^1\)

Daniel K. Park
dkd.park@yonsei.ac.kr

Teresa B. Ludermir
tbl@cin.ufpe.br

Wilson R. Oliveira
wilson.oliveira@ufrpe.br

Francesco Petruccione
petruccione@sun.ac.za

Adenilton J. da Silva
ajsilva@cin.ufpe.br

1 Centro de Informática, Universidade Federal de Pernambuco, Recife, Pernambuco 50740-560, Brazil

2 Department of Applied Statistics, Yonsei University, Seoul 03722, Republic of Korea

3 Department of Statistics and Data Science, Yonsei University, Seoul 03722, Republic of Korea

4 Departamento de Estatística e Informática, Universidade Federal Rural de Pernambuco, Recife, Pernambuco, Brazil

5 School of Data Science and Computational Thinking, Stellenbosch University, Stellenbosch 7600, South Africa

6 National Institute for Theoretical and Computational Sciences (NITeCS), Stellenbosch 7600, South Africa

7 Quantum Research Group, University of KwaZulu-Natal, Durban 4001, South Africa