The Journal of Supercomputing (2025) 81:354
https://doi.org/10.1007/s11227-024-06836-w

Efficient quantum circuit contraction using tensor decision diagrams

Vicente Lopez-Oliva[1] · Jose M. Badia[1] · Maribel Castillo[1]

Accepted: 17 December 2024

Abstract
Simulating quantum circuits efficiently on classical computers is crucial given the limitations of current noisy intermediate-scale quantum devices. This paper adapts and extends two methods used to contract tensor networks within the fast tensor decision diagram (FTDD) framework. The methods, called iterative pairing and block contraction, exploit the advantages of tensor decision diagrams to reduce both the temporal and spatial cost of quantum circuit simulations. The iterative pairing method minimizes intermediate diagram sizes, while the block contraction algorithm efficiently handles circuits with repetitive structures, such as those found in quantum walks and Grover's algorithm. Experimental results demonstrate that, in some cases, these methods significantly outperform traditional contraction orders like sequential and cotengra in terms of both memory usage and execution time. Furthermore, simulation tools based on decision diagrams, such as FTDD, show superior performance to matrix-based simulation tools, such as Google tensor networks, enabling the simulation of larger circuits more efficiently. These findings show the potential of decision diagram-based approaches to improve the simulation of quantum circuits on classical platforms.

Keywords Quantum circuit simulation · Tensor decision diagrams · Tensor networks · Contraction ordering methods · Quantum computing

✉ Vicente Lopez-Oliva
voliva@uji.es

Jose M. Badia
badia@uji.es

Maribel Castillo
castillo@uji.es

1 Departamento de Ingeniería y Ciencia de Computadores, Universitat Jaume I de Castelló, Avda. Sos Baynat, s/n, 12071 Castellón de la Plana, Castellón, Spain

1 Introduction

In recent years, quantum computing has captured the interest of researchers across a wide range of fields due to its potential to advance disciplines such as chemistry [1], pharmacology [2], and machine learning [3, Cap 1], among others [4]. However, currently and in a near future, we will only have noisy intermediate-scale quantum computers (NISQ) [5]. As a result, it is only feasible to execute algorithms with a limited number of qubits and gates before accumulating significant errors that render the results highly unreliable. Therefore, developing efficient simulators of quantum computers on classical computers is crucial for designing, validating, and improving quantum algorithms, as well as for designing and testing new quantum computers.

The standard method for simulating a quantum circuit involves breaking it down into a series of matrix–vector and matrix–matrix multiplications. As the number of qubits increases, the size of these vectors and matrices grows exponentially, leading to a rapid increase in the time needed for these operations. Despite that, the simulation of medium-sized quantum algorithms is mainly limited by the available memory, due to the substantial amount of space required to store the quantum states and operators [6, 7]. To address this issue, different techniques have been developed for circuit simulation, including full wave-function evolution [8], Feynman paths [9], and tensor network contraction [10]. Simulators based on tensor networks have demonstrated to be highly effective in simulating random quantum circuits (RQCs) [11]. However, their efficiency is highly dependent on the order in which the tensors are contracted. Determining the optimal order is an NP-hard problem [1] [12], so it is important to develop heuristics to efficiently approximate the optimal order.

In addition, alternative ways of representing, storing, and manipulating states and operations have been used to reduce the computational and spatial cost of simulation. For example, decision diagrams (DD) [13] are gaining relevance for the representation of quantum information due to their two main advantages. On the one hand, quantum circuits and operators can exploit mathematical properties to reduce data storage requirements. On the other hand, DDs store operations and take into account the structure of quantum operators to avoid their repetition [14]. Among the most notable variants of DDs are the quantum multiple-valued decision diagram (QMDD) [14], which provides a very efficient implementation, and tensor decision diagrams (TDDs), which combine the advantages of tensor networks and DDs [15].

In this study, we utilized the fast tensor decision diagram (FTDD) tool to implement our algorithms and perform the experiments [16]. This tool provides an efficient implementation of the TDD, enhanced with various optimizations. Specifically, we adapted and extended two methods employed to improve contraction ordering for quantum circuit simulation. These enhancements not only expand the functionality of the FTDD but also demonstrate distinct advantages depending on the circuit structure. The results highlight the potential of these methods when compared

[1] As a reminder, NP-hard problems are those that are at least as complex as NP problems, and are therefore in the group of higher complexity problems.

with existing ordering techniques used in this framework and other simulation tools based both on matrix and decision diagram representations.

The main contributions of this work are the following:

- Adaptation and implementation of two tensor network contraction ordering methods for simulating quantum circuits–iterative pairing and block contraction–within the FTDD framework. They are designed to optimize the contraction process for the TDDs used by the tool.
- Enhancement of the functionality of the block contraction method by adding a preprocessing method to automatically detect repeated subcircuits whose contraction can be avoided.
- Comprehensive evaluation of the proposed methods on various quantum circuits, demonstrating significant improvements in both temporal and spatial efficiency compared to traditional contraction orders such as sequential and cotengra.
- Exploitation of the repetitive structure of quantum circuits, such as quantum walks and Grover's algorithm, to reduce the computational and spatial cost of simulation.
- Comparative analysis of FTDD with other state-of-the-art quantum circuit simulation tools, highlighting the superior performance and scalability of decision diagram-based approaches.
- Experimental evaluation of the effect of contracting different implementations of the same quantum algorithm.

The paper is structured as follows: Sect. 2 briefly reviews some related work. Section 3 summarises the theoretical background of our work, including basic aspects of quantum computation, tensor networks, and decision diagrams. In Sect. 4, we detail the implementation of the contraction methods on the FTDD tool. In Sect. 5, we provide a detailed description of the experiments conducted and discuss the results. Finally, Sect. 6 presents the conclusions of the study and suggests directions for future research.

2 Related work

There are many quantum simulators that operate both sequentially and in parallel [17]. A survey of the main types of existing simulators can be found in [18]. Most of these quantum simulators evolve the full quantum state using matrices. Some of the most well-known simulators are IBM Qiskit, Google Cirq, QuEST [19], or qHiPSTER [20]. However, the exponential growth of the spatial and temporal cost of simulation restricts the number of circuit qubits to a few tens, even on the most powerful supercomputer available today.

Markov and Shi first proposed to use tensor networks to simulate quantum circuits [21]. Following this paper, a number of simulators based on tensor networks have been developed, such as qFlex [22], QuantEx [23], Jet [24], QTensor [25], TensorCircuit [26], or the one presented by Huang [27]. (An improved version of the one presented by Gray [7] called cotengra.) One of the most widely used simulators of this kind is quimb [28], which has been extensively utilized to obtain contraction orders that can challenge the quantum supremacy demonstrated by Google.

One of the most common approaches to order the contractions of tensor networks is based on the tree decomposition of the line graph associated with the network. It includes methods such as QuickBB [29] and Flowcutter [30]. Another approach is the hypergraph-based method, used for example in cotengra [31]. This method builds contraction trees using hypergraph partitioning such as KaHyPar [32]. In [7], the authors introduce several new heuristics to find good contraction paths and combine them with other well-known methods in a framework. They find that the orders obtained can be very close to the optimal. Also, they compare six methods on different tensor networks, some of them associated to quantum circuits. These methods include the exhaustive optimal ordering search found in in opt_einsum [33], QuickBB, and Flowcutter; one method based on hypergraph partitioning and another based on community detection and a greedy agglomerative method.

All of the aforementioned tools utilize matrix representations to perform operations. However, there are alternative techniques for handling quantum circuits. DD was initially proposed to represent switching circuits [34]. Following this work, DDs were adopted to represent data and operations for various applications, due to their ability to reduce both temporal and spatial costs. The first attempt to adapt this data structure for quantum computation was made in [35], where it was used to simulate a quantum circuit [13, 36]. This adaptation is known as reduced ordered binary decision diagram (ROBDD). Since then, many tools have been developed using this type of diagram. The most well-known library that implements this concept is MQT [14], which is utilized for both simulating and verifying quantum circuits. This tool implements a type of DD called QMDD [14].

Another type of DDs that combines tensor networks to represent quantum circuits and DD to store and manipulate them are the TDD, firstly introduced at [15]. This work focuses on the use of FTDD, an optimized implementation of TDDs introduced at [16].

3 Background

This section briefly discusses the basic theoretical foundations of quantum computing, tensor networks, and decision diagrams, which form the basis for the development of this work. For a more in-depth information on quantum computing and its mathematical foundation, you can refer for example to [3] or [37].

3.1 Quantum circuits as tensor networks

The basic unit of information in quantum computing is the qubit, which is the counterpart to the classical bit. The state of a qubit can be written as a linear combination of two basic states as $\alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$ [38]. A quantum system is composed of a set of n qubits, $\phi_1, \cdots, \phi_n$, described by a Hilbert space of dimension 2^n, whose basic states are in the set $\{0,1\}^n$. Therefore, the state of a quantum system $|\phi\rangle$ can be described as:

$$|\phi\rangle = \sum_{i\in\{0,1\}^n} \alpha_i |i\rangle \text{ with } \forall i \in \{0,1\}^n, \alpha_i \in \mathbb{C}, \tag{1}$$

where the probability of measuring a particular state $|i\rangle$ is given by $|\alpha_i|^2$. Quantum systems are manipulated through quantum operators, represented by unitary matrices of dimension $2^n \times 2^n$. Typically, these operators only act on a subset of the system's qubits. Quantum circuits are commonly used to represent the successive application of quantum operators $g_1, \cdots, g_k$ to a quantum state. Quantum algorithms can be represented by a quantum circuit. To relate input states to their corresponding output states, a unitary operator representing the complete functionality of the circuit can be constructed.

On the other hand, tensors are a natural generalization of vectors and matrices. A tensor of rank r is defined as an element $d_1 \times \cdots \times d_r$ existing in the space $\mathbb{C}^{d_1,\cdots,d_r}$. Then, a complex value can be seen as a tensor of rank 0, while a vector with d complex elements (dimension d) is a tensor of rank 1. Similarly, a matrix of dimension $n \times m$ is a tensor of rank 2 in $\mathbb{C}^{n\times m}$ space. Tensors have also become popular due to their simple notation and graphical representation, which facilitates the visualization of tensor networks. Tensors have a defined set of operations, including the tensor product, trace, contraction, and partitioning [39]. One of the most significant operations that can be performed on tensors is the contraction. The contraction of two tensors is a tensor obtained by summing up over the shared indices. To exemplify, if two tensors, $R_{x,z}$ and $S_{y,z}$, are considered, with the common index z, the resulting contraction, $T_{x,y}$, is defined as follows:

$$T_{x,y} = \sum_{z\in\{0,1\}} R_{x,z} \cdot S_{y,z} \tag{2}$$

In general, the successive contraction between pairs of tensors of a network allows to obtain a single tensor representing its full functionality.

The quantum state $|\psi\rangle$ of a qubit can be represented as a vector

$$|\psi\rangle = [\alpha_0, \alpha_1]^T, \text{ where } \alpha_0, \alpha_1 \in \mathbb{C}. \tag{3}$$

Therefore, it can be represented as a rank 1 tensor T_q. Similarly, the operators associated with the gates applied to a single qubit are matrices of size 2×2, and can be represented as rank 2 tensors T_{q_1,q_2}. In general, a gate acting on n qubits can be represented as a tensor of rank $2n$. Note that in tensor notation, input and output indices are not distinguished.

A quantum circuit can be modeled as a tensor network, where each gate is represented as a tensor, and the lines connecting the qubits between gates serve as shared indices between these tensors. Consequently, since the behavior of a quantum circuit can be simulated using the matrix representing its functionality, it can similarly be simulated using its tensor representation. This implies that simulating a quantum circuit is equivalent to contracting its associated tensor network.

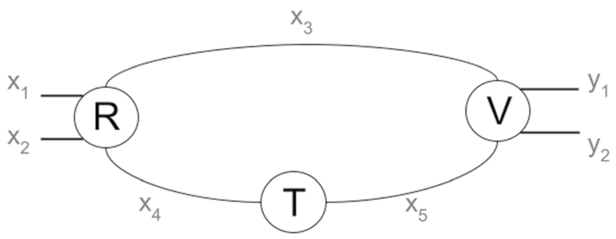


Fig. 1 Example of a tensor network with 3 tensors. It has 3 closed indices (x_3, x_4, x_5) and 4 open indices (x_1, x_2, y_1, y_2)

This principle underpins the approach proposed by Markov and Shi to reduce the cost of simulating quantum circuits [21]. They demonstrated that if the *treewidth* of the underlying graph associated with the tensor network is logarithmic in the number of gates, the quantum circuit can be simulated with polynomial cost in the number of gates. Furthermore, the *treewidth* aligns with the maximum rank of all tensors encountered during the contraction process and is a key factor in determining the overall simulation cost.

3.2 Contraction ordering methods for tensor networks

The goal of ordering methods is to identify a contraction sequence that approximates the optimal order, thereby minimizing the temporal and spatial costs of the contraction process. These costs are heavily influenced by the rank of the intermediate tensors produced during contractions, higher ranks leading to exponentially increased costs. For instance, consider the tensor network illustrated in Fig. 1. If we start by contracting R with T, the resulting tensor will have four indices (x_1, x_2, x_3, x_5). On the other hand, if we first contract the tensors R and V, the resulting tensor will have six indices $(x_1, x_2, x_4, x_5, y_1, y_2)$, resulting in a higher cost due to the larger rank of the intermediate tensor. Thus, selecting an effective contraction order involves avoiding scenarios that lead to high-rank intermediate tensors, thereby reducing the overall contraction cost.

Obtaining a good contraction order has been extensively studied, and various methods and heuristics have been proposed, as we have seen in Sect. 2. One of the most significant proposals is based on the results of Markov and Shi [21], which obtains a tree decomposition of the *line graph* associated with the tensor network. Other proposals are based on the use of greedy methods, which use heuristics to choose the next pair of tensors to contract based on the current state of the network. Different strategies can be employed such as selecting the two tensors that share the most indices or the two tensors with the lowest rank. The iterative pairing method that we have implemented using TDDs falls within this family of methods.

3.3 Decision diagrams

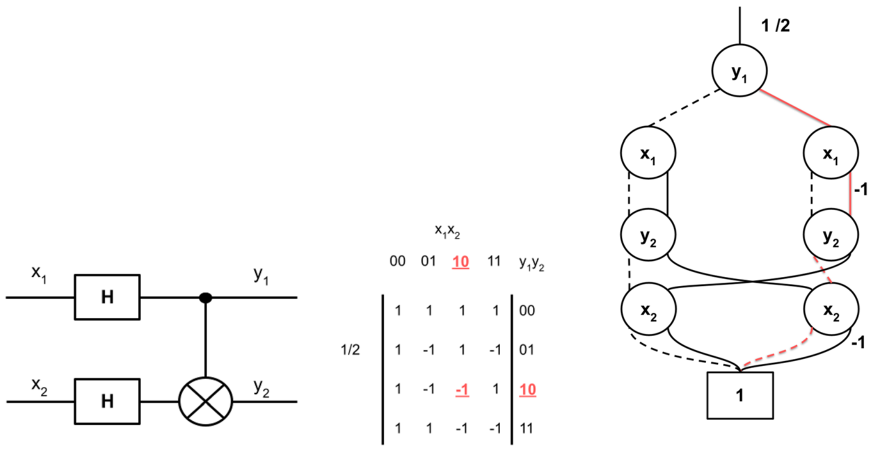
Quantum circuits are usually implemented by using vectors and matrices. Its main advantage is the availability of highly efficient algorithms for matrix multiplication in all types of architectures, whether sequential or parallel. However, this is not the only type of structure that we can use to implement them. DDs are acyclic undirected graphs that represent the gates that compose the circuits and the unitary matrices used during its processing (simulation, verification, etc.). To accomplish this, the matrix is divided into a predetermined number of sub-matrices based on a specific criterion.

Two main data structures are usually employed to efficiently store the DDs and the operations in order to reduce the spatial and temporal cost. On the one hand, the `unique table` stores the information about the graph nodes in such a way that nodes with identical information are not duplicated, thus significantly reducing the storage cost. On the other hand, the `compute table` stores the operations as they are performed, so that they are not repeated when they are needed again. This implementation is very advantageous because the operations on tree-like DDs are usually performed recursively based on the subtrees of each node [15]. In many cases, these subtrees are repeated, both in the construction of a diagram and in the contraction of two of them. If an operation between subtrees has already been performed, it is retrieved directly from the compute table, avoiding repetition. The affected subtrees can be large, which would avoid a high percentage of computations with DDs.

Tensors and tensor networks can also be represented as DDs using TDDs [15]. This allows the benefits of both techniques to be leveraged in the handling of quantum circuits. In order to achieve this objective, a number of methods have been developed that are essential for DDs to be able to work with tensor networks. These methods are recursive and are based on the traversal of tree-like DDs. Typically, when utilizing these operations, we work with reduced and normalized versions of the DDs, as this enables us to exploit the benefits of spatial and temporal advantages. One of the most crucial operations that must be implemented and falls within this category is the contraction. The implementation details of this operation can be found in [15].

TDDs use nodes to represent the indices of the tensor network, with each node having two successors linked by edges. One of the successors is associated with the value 0 of the node (which is usually represented by a dashed line), while the other successor is associated with the value 1 of the node (which is usually represented by a solid line). Each edge has a weight indicating the value by which the TDD represented by the successor must be multiplied. In the graphical representation, the absence of a weight on the edge indicates that its weight is 1.

Figure 2 shows a simple example of how to represent a circuit as a TDD. The circuit shown on the left of the Fig. 2a can be represented by the matrix on the right, which defines its functionality. Using this matrix, we can construct the TDD as shown in Fig. 2b. If all the elements of the matrix were unique and without common factors, the resulting diagram would be a tree with four levels of nodes, one per index, and a final level with the 16 values of the matrix. TDDs leverage



(a) Example of a quantum circuit and its associated unitary matrix represented by a tensor

(b) TDD representing the circuit. The path defined by the edges (in red).

Fig. 2 Comparison of the matrix and TDD representations of a quantum circuit

the structure and repetition of elements or blocks in matrices to reduce the number of nodes and edges required to store and operate with tensor networks.

Another well-known type of DD is the QMDD [40], which provides a compact representation of the matrix associated with the circuit functionality. QMDDs are constructed by recursively dividing the matrix into four quadrants. This results in a quaternary tree in which the leaves correspond to the elements of the matrix. Normalization and reduction techniques based on the structure and repetitions in the sub-matrices are applied in order to obtain a compact and canonical representation of the functionality of the quantum circuit. The main differences between QMDD and TDD are that the former is based on matrix multiplication instead of tensor contraction, and that the depth levels in QMDD represent qubits instead of indices as in TDD. Furthermore, QMDDs can be transformed into TDDs in a natural and flexible way, since the order of the indices does not have to respect the order of the qubits. This provides TDDs with advantages for certain circuit types that heavily utilize SWAP gates, as it can be seen in [15] and [41].

The TDD introduced in [15] is implemented in Python and do not include the majority of the implementation-level improvements that are included in the tools that implement QMDD. As a consequence, in place of the original TDD tool, the improved C-implemented version of this tool, designated FTDD, will be employed [16]. In this new version, the authors implemented in C++ the TDD with the contraction orders `sequential` and `cot` and the aforementioned improvements. That includes improvements such as edge-centric data structures for TDD, employment of key BDD optimizations, or the execution of near-optimal contraction order (`cot` in this case), with which the authors can obtain speedups up to 175 respect to the python version.

One of the improvements that FTDD implements is the Tetris preprocessing. Tetris is a rank simplification technique that is applied prior to getting the contraction order to reduce the number of gates to be simulated by consolidating tensors in a local manner. The algorithmic process is analogous to that of the renowned video game from which it derives its nomenclature. To accomplish this, the initial step is to place the first tensor corresponding to the initial gate identified in accordance with the specified order of the quantum circuit on each line of qubits. Subsequently, the gates of the circuit are methodically examined and treated as if they were descending along the lines of qubits, akin to the manner in which tetrominoes are arranged in the game of Tetris. Each gate is then examined in relation to the top tensors of the stacks corresponding to the associated qubits. In the event that there are shared qubits between the gate and the top tensors, and the rank simplification constraint is satisfied, the gate is consolidated with the top tensors by a contraction, with the resulting tensor replacing the top tensors. In the event that this is not the case, the gate is moved to the uppermost positions within the stacks.

4 Contraction ordering method proposals

In this section, we will describe two contraction ordering methods based on the proposals described in [42], where they were used in an equivalent problem for ordering matrix multiplications with QMDDs. Both methods aim at exploiting the properties of DDs to reduce the number and cost of intermediate contractions needed to obtain the final tensor representing the circuit functionality.

4.1 Iterative pairing

The most straightforward method for ordering contractions in a tensor network involves traversing the gates of the circuit and sequentially contracting them with the resulting accumulated diagram. We will call this the `sequential` method. However, with the multiplication of quantum operations, their representations become increasingly complex, thus reducing their sparsity, increasing the level of entanglement between qubits and eliminating existing redundancies, which in turn increases the cost. In other words, the bottleneck arises from the consequences of performing successive multiplications.

In [42], a solution to a similar problem using QMDDs is proposed. The objective is to derive the matrix representing the complete functionality of the circuit by multiplying the matrices associated with each gate. The authors suggest taking advantage of the associative property of matrix multiplication to combine matrices associated with disjoint pairs of gates. Initially, we multiply the matrices of individual gates in the circuit. Then, we iteratively combine pairs of matrices obtained from the previous iteration until the final matrix is produced. This method (called from now on the `iterative` method) allows for better exploitation of redundancies and results in DD with smaller average sizes compared to those obtained by cumulative multiplication, as demonstrated in the paper. This is because, by dividing the multiplications in this way, the representations are kept as small as possible. The proposed pairing of multiplications is as follows:

$$U = (U_1 \cdot U_2) \cdot (U_3 \cdot U_4) \cdot \ldots \cdot (U_{n-1} \cdot U_n), \tag{4}$$

where U is the matrix representing the functionality of the circuit, and U_i is the matrix representing the functionality of the gate i. After applying the first iteration of the algorithm, the following equation is obtained:

$$U = U_{1,2} \cdot U_{3,4} \cdot \ldots \cdot U_{n-3,n-2} \cdot U_{n-1,n} \tag{5}$$

The algorithm proceeds to the next step to obtain the next level of matrices. In this step, the same associative property is applied again to that set of matrices, performing the same type of matching between them. This process can be applied iteratively to each of the successive levels obtained until a single matrix is calculated. Assuming, for simplicity, that the number of gates in the circuit n can be expressed as $n = 2^k$ for a certain $k \in \mathbb{N}$, the proposed scheme defines a tree-like pairing order.

This approach can also be applied to contract the tensor network representing a quantum circuit. At each step, all disjoint tensors at the same level are contracted in pairs, resulting in a new set of tensors for the next level. This process is repeated until a single tensor remains. The initial level consists of tensors representing the gates of the quantum circuit, and the final tensor represents the complete functionality of the circuit.

4.2 Block contraction algorithm

The block contraction algorithm (named `blocks` from now on) is based on the approach proposed in [42] to exploit the repetition of gate groups that occurs in many quantum circuits, such as those implementing the Grover's search algorithm [43], quantum random walk [44], amplitude estimation [45], and quantum phase estimation [46].

This type of circuits typically begins with an initialization phase consisting of a set of gates represented by the DD $U\_\{\text{in}i\}$. This is followed by several iterations of blocks of identical gates, where each block is represented by the diagram U_{iter}, and a set of final gates represented by U_{end}. The DD U that represents the full functionality of the quantum circuit can then be obtained as:

$$U = U_{\text{end}} \cdot U_{\text{iter}} \cdot \ldots \cdot U_{\text{iter}} \cdot U_{\text{in}i} = U_{\text{end}} \cdot U_{\text{iter}}^N \cdot U_{\text{in}i} \tag{6}$$

where we assume for simplicity that $N = 2^k$ for $k \in \mathbb{N}$.

This quantum circuit structure enables the use of a multistep ordering method. First, the iterative pairing method proposed in the previous section is used to obtain the DDs $U\_\{\text{in}i\}$, U_{iter} and U_{end}. Subsequently, all the diagrams representing the iterations, U_{iter}, are contracted to obtain U_{iter}^N. To perform this operation, we also apply the iterative pairing method as follows.

$$\begin{aligned} U_{\text{iter\_final}} &= U_{\text{iter}} \cdot U_{\text{iter}} \cdot \ldots \cdot U_{\text{iter}} = (U_{\text{iter}}^{2^0} \cdot U_{\text{iter}}^{2^0}) \cdot \ldots (U_{\text{iter}}^{2^0} \cdot U_{\text{iter}}^{2^0}) \\ &== (U_{\text{iter}}^{2^1} \cdot U_{\text{iter}}^{2^1}) \ldots (U_{\text{iter}}^{2^1} \cdot U_{\text{iter}}^{2^1}) = \ldots = U_{\text{iter}}^{2^k}, \end{aligned}$$

The main advantage of the `blocks` algorithm arises from the fact that at each pairing level, from $l = 0$ to $k - 1$, we only need to perform one contraction $U_{\text{iter}}^{2^l} * U_{\text{iter}}^{2^l} = U_{\text{iter}}^{2^{l+1}}$, since the rest will be exactly the same and can be avoided by reusing the results stored in the compute table. As a last step of the `blocks` algorithm, we will calculate the DD that represents the full functionality of the circuit:

$$U = U_{\text{end}} \cdot U_{\text{iter\_final}} \cdot U_{\text{in}i}$$

The original article [42] presents this idea with a significant drawback: it requires prior knowledge of the circuit type to identify its constituent blocks. In other words, the algorithm needs to be provided with the blocks of operations included in the circuit. This limitation prevents users from taking advantage of the method for circuits that have blocks but whose structure is unknown or is not programmed into the tool. To address this, we propose incorporating an algorithm that begins by searching for possible blocks within a circuit.

The algorithm takes a file in quantum assembler format (qasm, [47]) as an input parameter. This file is processed gate by gate checking for identical operations done in the same set of qubits and in the same order. Then, it tests whether at least one block of gates (operations) is repeated. This happens when all the gates between the first gate of the block and its next occurrence are exactly the same as the following ones, and acts in the same set of qubits in the exact same order. If there are two consecutive identical blocks, the algorithm tries to find more repetitions of the same block until it reaches a nonidentical block. The size and total number of gates in each block are recorded. This process is repeated for each gate in the file, and the configuration containing more gates in the blocks is returned.

Listing 1 presents the algorithm used to perform block contraction after identifying the repeated blocks. The method takes as input a list of tensors representing the circuit, parameters for block detection, and the contraction method to be applied. The required block detection parameters include the number of gates in $U\_\{\text{in}i\}$, U_{iter} and U_{final} (lines 3-5). Using this information, the algorithm determines the start and end of each block and the number of repetitions (lines 6 and 7). The blocks are contracted from lines 11 to 16. Note that the calculation of U_{iter} is not repeated, thanks to the use of the `compute  table` in decision diagrams, which avoids redundant calculations. After contracting the blocks, in line 18, we apply the iterative pairing ordering method to contract the resulting tensors. This algorithm takes into account scenarios where the condition $N = 2^k$ is not satisfied, handling cases such as $U_{\text{iter}}^2 * U_{\text{iter}}$.

```
def blocks(tensors, args, contraction):
    n = len(self.tensors)
    n_ig = args.n_init_gates
    n_gxb = args.n_gates_x_block
    n_lg = args.n_last_gates
    n_tensor_blocks = (n - n_ig - n_lg) // n_gxb
    tensor_blocks = [None for _ in range(n_tensor_blocks)]
    i_g = n_ig
    i = 0
    # The contractions of all blocks are performed
    while i_g < n - n_lg:
     i_lg = i_g + n_gxb
     # The contraction of each block is performed
     tensor_blocks[i] = contraction(tensors[i_g:i_lg])
     i += 1
     i_g = i_lg
    # The iterative pairing method is used to contract the blocks.
    tdd = iterative_pairing(tensor_blocks)
    # Initial gates are contracted on the result of the blocks
    if n_ig > 0:
     tdd = cont(contraction(tensors[:n_ig]), tdd)
    # The final gates are contracted on the result of the blocks.
    if n_lg > 0:
     tdd = cont(tdd, contraction(tensors[-n_lg:]))
  return tdd
```

Listing 1: Block contraction algorithm.

The algorithm in Listing 1 requires prior knowledge of the blocks within the circuit and the number of gates in each block, making it necessary to preprocess the circuit to gather this information. Listing 2 outlines this preprocessing step. The algorithm takes a file describing the quantum circuit in quantum assembler format (qasm) as its input parameter [47]. It processes the file gate by gate (line 9), checking for identical gates (line 17). It then verifies whether at least one block of gates is repeated (lines 21 and 24), identifying this scenario when all the gates between the first occurrence of the block and its next occurrence are exactly the same as the subsequent ones (line 24). If two consecutive identical blocks are detected, the algorithm attempts to find further repetitions of the same block until it encounters a non-identical block (line 28). The size of each block and the total number of gates it contains are then recorded (lines 32-37). This process is repeated for each gate in the file, ultimately returning the configuration with the largest number of gates within the blocks (line 42).

```python
def get_blocks_params(file):
 circ = read_file(file)
 N = len(circ)
 n_gates_ini = N
 n_gates_final = 0
 block_len = 0
 max_gates = 0
 # Run through all the doors of the file
 for i in range(N):
  i_final = i + 1
  isFinal = False
  # We check for all the identical doors that we can find in the file.
  while not isFinal:
   # The try block is necessary because searching for the index of an element will
       result in an error if the element is not found.
   try:
    # Search for the next appearance of the gate i
    j = circ.index(circ[i], i_final)
    i_final = j + 1
    i_len = j - i
    # If there are not enough gates to complete a block, move on to the next gate.
    if i + i_len >= N:
     continue
     # Consecutive gates are checked to see if they are identical.
    if circ[i:j] == circ[j:(j + i_len)]:
     k_ini = j
     k_fin = j + i_len
     # Try to find as many consecutive blocks as possible
     while k_fin <= N and circ[i:j] == circ[k_ini:k_fin]:
      k_ini += i_len
      k_fin += i_len
     # k_ini will contain the number of the last gate of the last block.
     n_gates = k_ini - i
     if n_gates > max_gates:
      max_gates = n_gates
      block_len = i_len
      n_gates_ini = i
      n_gates_final = N - k_ini
   except ValueError:
    isFinal = True
   # It stops searching if it cannot find blocks that cover more gates.
   if N - i < max_gates:
    break
 return [n_gates_ini, block_len, n_gates_final]
```

Listing 2: Algorithm for detecting repeated blocks in a quantum circuit.

It is important to note that contraction ordering algorithms such as the sequential method, the iterative pairing, or the block contraction algorithm described in this section rely on traversing the circuit's gates stored in a specific sequence. Altering this sequence may change the contraction order obtained by the algorithms and could significantly affect the cost of the contraction process.

5 Circuit simulation and results

This section describes the methodology used to perform the experiments, as well as the results obtained and their analysis. Our goal is to compare the temporal and spatial cost of the two proposed ordering methods with others present in the literature for a varied set of well-known quantum circuits using different tools.

5.1 Experimental environment and methodology

The experimental analysis was carried out using a server with 4 Intel Xeon Gold 6330 H processors at 2 GHz with 24 cores each, a total of 394 GiB of DDR4 RAM and 33 MiB of cache memory. We used the Linux 5.4.0-72-generic 80-Ubuntu operating system.

In order to evaluate the efficiency of the methodologies outlined in the preceding sections, a variety of tools and circuits will be used. The tools will be utilized in the subsection 5.2.3 to assess their relative capabilities. With regard to the circuits to be utilized, there are two main categories: those comprising repeating blocks (QWalk and Grover) and those devoid of repeating blocks. The latter will be employed in the subsection 5.2.1. The selected circuits are associated with some of the most well-known quantum algorithms and are commonly utilized not only with tensor network methods [15, 16], but also with matrix representations of quantum circuits [48, 49] and various types of DD [41, 50]. Most of the circuits can be found at [51].

It is important to note that all simulators used in our experiments can be configured to perform different types of simulations. The computational and spatial cost varies depending on the type of simulation performed. The main categories of circuit simulations are as follows:

- **Unitary simulation**: Computes the unitary matrix that represents the complete functionality of the quantum circuit. This corresponds to contracting the tensor network associated with the circuit while keeping both the input and output indices open.
- **Statevector simulation**: Computes the evolution of the entire quantum state as it propagates through the quantum circuit for a given input state (typically $|0\rangle^n$). This corresponds to contracting the tensor network with the input indices closed and the output indices open.
- **Amplitude simulation**: Computes the probability amplitude for a specific input–output state pair. This corresponds to contracting the tensor network with both the input and output indices closed.

Table 1 provides a comprehensive overview of the metrics employed for the circuits utilized in the experiments conducted in this study. We use several metrics that defines the circuit, including the number of qubits (referred to as “n"), the total number of gates (referred to as “# gates"), the number of one-qubit gates (referred to as “# gates 1qb"), the percentage of one-qubit gates (referred to as “% gates 1 qb"), the total number of indexes associated with all tensors in the tensor network when the circuit is open (referred to as “# indexes"), and the depth of the circuit (referred to as “depth"). As evidenced by the table, the circuits exhibit a considerable degree of diversity. Notably, the values of the metrics are much larger for the circuits with blocks (QWalk and Grover) for a similar number of qubits. It is also noteworthy that the "% gates 1qb" parameter of the circuit exhibits the greatest variability among the selected circuits, with percentages ranging from a low of 7% to a high of 76%. In general, this percentage tends to decrease as the number of qubits in the circuit

Table 1 Metrics of the circuits that will be used in the following tests

Circuit	n	# gates	# gates 1qb	% gates 1 qb	# Indexes	Depth
QAOA [52]	9	45	27	60.00%	135	9
	10	50	30	60.00%	150	10
	11	55	33	60.00%	165	9
	12	60	36	60.00%	180	10
VQE [53]	13	63	39	61.90%	187	17
	14	68	42	61.76%	202	18
	15	73	45	61.64%	217	19
	16	78	48	61.54%	232	20
QFT [46]	18	180	18	10.00%	702	36
	20	220	20	9.09%	860	40
	22	264	22	8.33%	1034	44
	24	312	24	7.69%	1224	48
QGAN [54]	10	65	20	30.77%	230	19
	11	77	22	28.57%	275	21
	12	90	24	26.67%	324	23
	13	104	23	25.00%	377	25
QNN [55]	8	223	104	46.64%	692	76
	9	278	126	45.32%	869	86
	10	339	150	44.25%	1066	96
	11	406	176	43.35%	1283	106
QPE [46]	12	346	202	58.38%	992	107
	13	400	231	57.75%	1151	110
	14	453	260	57.40%	1306	113
	15	531	301	56.69%	1537	142
QWalk [44]	9	23796	11964	50.28%	71265	19652
	11	97428	48796	50.08%	292131	80932
	13	392244	196220	50.02%	1176549	326532
	15	1178847	589509	50.00%	3536385	982059
Grover [43]	5	1461	765	52.36%	4319	1205
	6	4282	2234	52.17%	12666	3429
	7	13715	6995	51.00%	40977	11141
	8	36780	18604	50.58%	109920	30213
RQC [56]	10	115	87	75.65%	302	11
	12	135	103	76.30%	350	13
	14	151	111	73.51%	398	15
	16	172	128	74.42%	448	17

increases. Table 1 also contains references for each type of circuit. In them, this can be found more details about the definition and applications of the circuits.

The number of qubits used in each circuit during the testing phase depends on the maximum number that could be simulated in our experiments. As the number of qubits increases, simulations become impractical due to either exhausting the total available memory or exceeding the preestablished time limit of 24 h. This limitation accounts for the reduced number of qubits presented in Table 1, as it reflects the maximum feasible size for which comparisons could be performed using unitary simulation-a method that is extremely demanding in both time and memory.

In the tables, the cells colored bold represent the best results for each row. The potential errors are labeled as follows:

- **Out of memory (O.M.)**: We run out of available memory and we could not fit the circuit in memory.
- **Time out (TimeOut)**: The simulation runs for more than 24 h.

5.2 Experimental results

5.2.1 Evaluation of different methods of ordering contractions

This section will present the simulation of six circuits, each of which exhibits a distinct structural configuration, according to the metrics included in Table 1. We evaluate the performance of 3 different ordering methods implemented in the FTDD tool, which are: the sequential method (named `sequential` from now on), the iterative method (named `iterative` from now on), and the order given by cotengra (named `cot` from now on). We will configure the tool to perform unitary simulations (that means, both the input and output indices will be open).

The aim is to find the best method for ordering the contraction of open circuits. This is because we want to find the best method for contracting each block of the `blocks` algorithm, where the contraction of each individual block can be considered as the contraction of a circuit with open indices.

Table 2 presents the execution time and memory usage for the simulations. Notably, for the `cot` method, this time is further divided into contraction and ordering time. The latter refers to the time taken to compute the order. This division between contraction and ordering time was considered useful, as in most experiments, ordering represented the majority of the cost. In contrast, for the other two methods, this step incurs no additional cost since the order is predetermined and does not require calculation. Thus, in these cases, the execution time corresponds solely to the contraction time. With regard to memory usage, it is evident that the order `cot` represents the optimal choice for almost all circuits and sizes. However, when considering the contraction time required, the `cot` is only the best for the QAOA circuit and without considering the ordering time. If we include the cost of obtaining the order, `cot` is only the fastest method in the two largest versions of this type of circuit. In this regard, the `iterative` and `sequential` methods are the fastest options in most cases. When comparing `sequential` and `iterative` methods exclusively, analysis of the % gates 1qb metric reveals that in instances where it falls below 30% (QFT and QGAN), the `iterative` method is preferable. Conversely, when it exceeds 40% (QAOA, VQE, QNN, and QPE), the `sequential` method emerges as the best strategy.

Table 2 Results of performing unitary simulations with FTDD using different contraction orders

Benchmarks		Execution time (s)				Memory usage (Mb)		
Circuit	*n*	`Sequential`	`Iterative`	`Cot`		`Sequential`	`Iterative`	`Cot`
				Contraction	Order			
QAOA	9	102.05	226.47	*58.32*	734.42	423.36	289.94	*195.29*
	10	36.63	36.32	*0.33*	722.34	374.05	170.61	*27.72*
	11	9,676.73	17,096.15	*3,645.01*	725.10	7,349.51	4,347.04	*2,091.57*
	12	**O.M.**	**O.M.**	*14,529.13*	740.34	**O.M.**	**O.M.**	*4,908.77*
VQE	13	23.21	44.87	*14.25*	715.76	299.63	383.48	*217.50*
	14	*276.08*	492.24	434.24	701.23	1,780.79	1,537.64	*1,550.66*
	15	22,069.35	*17,613.46*	**O.M.**	**O.M.**	15,331.52	*9,314.90*	**O.M.**
	16	*454.69*	7,694.27	**O.M.**	**O.M.**	*2,728.53*	8,650.53	**O.M.**
QFT	18	3,717.24	*64.49*	95.47	3,600.74	2,242.73	2,312.71	*580.91*
	20	**TimeOut**	*217.04*	1,181.72	3,605.38	**TimeOut**	6,808.48	*3,628.43*
	22	**TimeOut**	*217.04*	1,181.72	3,605.38	**TimeOut**	6,808.48	*3,628.43*
	24	**TimeOut**	*2,688.44*	10,556.57	3,613.52	**TimeOut**	52,054.11	*6,950.88*
QGAN	10	110.03	*69.41*	107.29	3,611.27	364.75	*223.52*	242.82
	11	1,422.12	*509.93*	544.11	3,611.83	1,696.73	*780.34*	814.96
	12	28,619.07	29,940.78	*4,901.78*	342.86	12,521.45	6,265.39	*2,794.70*
	13	**TimeOut**	*22,016.52*	35,206.80	3,610.53	**TimeOut**	*6,982.79*	7,581.86
QNN	8	*8.33*	949.77	14.08	265.24	140.25	330.80	*110.75*
	9	*133.81*	9,722.81	422.33	264.35	591.24	1,650.57	*586.27*
	10	*1,628.87*	**TimeOut**	4,212.08	271.30	2,503.12	**TimeOut**	*2,495.95*
	11	*15,467.34*	**TimeOut**	**O.M.**	**O.M.**	*10,582.41*	**TimeOut**	**O.M.**
QPE	12	*168.04*	1,429.71	301.41	773.91	1,502.31	1,257.14	*749.88*
	13	*1,562.13*	23,107.16	2,491.92	2,058.88	6,186.98	5,776.72	*2,964.72*
	14	*6,126.27*	**TimeOut**	8,587.96	1,411.76	17,873.52	**TimeOut**	*7,891.84*
	15	**TimeOut**	**TimeOut**	*40,528.28*	1,191.91	**TimeOut**	**TimeOut**	*29,640.93*

In order to get a global assessment of the efficiency of the different ordering methods, we have computed the median speedup of the `iterative` and `cot` methods with respect to the `sequential` method. In Table 2, we compute the median of the speedups for all sizes for each type of circuit. Only results where the 3 methods performed the final contraction were used to calculate the speedups. Note also that the speedups of the `cot` methods were calculated using only the contraction time. Table 3 shows that in most cases there are not large differences between the median contraction times using the three ordering methods. The best method depends on the type of circuit. For example, the `iterative` method has a much better performance for the QFT circuits, but a very poor performance for the QNN circuits.

Table 3 Speedups of performing unitary simulations with FTDD using different contraction orders with respect to the `sequential` method

Circuit	`Iterative`	`Cot`
QAOA	0.981	1.750
VQE	0.910	1.120
QFT	16.933	3.500
QGAN	1.629	0.987
QNN	0.056	0.687
QPE	0.822	0.942
Median	0.946	1.053

5.2.2 Exploitation of structural repetitions

In this section, we will assess the efficiency of the FTDD tool using the `blocks` method and compare it with the three ordering methods evaluated in the previous section. For the experiments, we use two types of circuits that include repetitions of large blocks of gates: QWalk and Grover. These repetitions arise from the fact that the circuits are associated to iterative algorithms where a part of the circuit is executed several times. In all cases, we have performed the simulations with circuits including four repeated blocks. For these experiments, we configure the tool to perform amplitude simulations (that is, both input and output indices are closed). This is a very common type of simulation, especially when dealing with very large or complex circuits, such as the RQC [22] or Sycamore circuits used in our experiments, or when comparing different contraction ordering methods with very large tensor networks [7]. It is also the worst case for the `blocks` method. Closing the input is advantageous for the other order contraction methods, since starting by contracting these indices results in a reduction of the range of tensors. However, this advantage is not exploited in the `blocks` method because the input qubits are contracted last to maintain the integrity of the block structure.

It is important to note that in all the methods employed by the FTDD tool, the Tetris optimization method described in [16] has been applied to simplify the tensor network represented by the circuit, with the exception of the `blocks` order. We do not apply Tetris or any form of preprocessing before the contraction process when using the `blocks` method, as doing so would modify the QASM representation of the circuit, which is essential for detecting the blocks. Furthermore, in most cases, applying Tetris can result in losing the block structure of the circuit. This implies that the results of the experiments will allow us to determine whether utilizing the block structure is more advantageous than any preprocessing that can be conducted prior to this stage. It is also worth noting that once the blocks are identified, an additional advantage can be gained by applying optimization techniques (preprocessing) to each individual block. One potential optimization that could be applied within each block is the recursive detection of smaller blocks, which should further enhance its performance. It should be noted, however, that these improvements have not been incorporated into the present work and are planned as future research.

Table 4 Execution times in seconds of performing amplitude simulations with FTDD using different contraction orders in circuits which have structural blocks

Circuit	n	`Sequential`	`Iterative`	`Cot`	`Blocks`
QWalk	9	487.33	478.02	471.37	*4.09*
	11	11037.17	11172.16	**O.M.**	*76.30*
	13	**O.M.**	**O.M.**	**O.M.**	*1491.17*
	15	**O.M.**	**O.M.**	**O.M.**	*36733.63*
Grover	5	0.77	0.76	0.77	*0.03*
	6	8.46	8.43	8.57	*0.20*
	7	122.84	135.21	124.91	*7.07*
	8	*1038.34*	6559.69	1031.92	11491.14

Table 4 presents the contraction times, in seconds, for the two circuit types with repeated blocks. Results show that exploiting the repeated blocks of the algorithm clearly reduces the temporal and spatial cost of the simulation, even if we do not use the simplifications provided by the Tetris technique. In the case of the QWalk circuits, which have the largest number of gates and indexes (see Table 1), the `blocks` method allows us to simulate larger circuits and is the fastest when all the methods finish the simulation. As for the Grover circuits, the `blocks` methods is again the fastest except for the circuit with 8 qubits.

5.2.3 Evaluation of different simulation tools

In this section, we will use the two repeated block circuits from Sect. 5.2.2 and one of the circuits from Sect. 5.2.1 to compare the behavior of different simulation tools. All tools were configured to perform amplitude simulations with identical input and output states. This approach was chosen for the same reason as above, namely that it allows the simulation of circuits with a larger number of qubits, in particular those belonging to the RQC category. The selected tools use different methods to represent quantum circuits, including matrix-based and DD-based approaches. The ordering methods evaluated were `cot` and `blocks` which are the most efficient methods or the only ones available in the tool. It is important to note that in circuits where no repeated block structures are detected, the `blocks` method is similar to the `iterative` method.

The tools used are the following:

- **Fast Tensor Decision Diagrams (FTDDs**[2]**)**: Simulation tool that uses TDDs to represent the information.
- **Google Tensor Networks (GTNs**[3]**)**: Tool for contracting tensor networks that uses matrices to represent the information.

[2] https://github.com/QiruiZhang/FTDD

[3] https://github.com/google/TensorNetwork

Table 5 Execution times of performing amplitude simulations using different contraction orders and tools for different types of circuits presented in seconds

Benchmarks		FTDD		quimb	GTN		DDSIM
Circuit	n	`Blocks`	`Cot`	`Cot`	`Cot`	`Iterative`	`Iterative`
QWalk	9	4.09	471.38	8.54	**O.M.**	507.89	*0.23*
	11	76.30	**O.M.**	**O.M.**	**O.M.**	11001.84	*0.72*
	13	1491.18	**O.M.**	**O.M.**	**O.M.**	**O.M.**	*2.73*
	15	36733.63	**O.M.**	**O.M.**	**O.M.**	**O.M.**	*8.03*
Grover	5	0.03	0.77	0.48	0.83	0.81	*0.04*
	6	0.20	8.57	1.44	8.70	10.18	*0.07*
	7	7.07	124.91	5.12	136.88	132.42	*0.35*
	8	11491.14	1031.92	16.28	1099.90	1078.32	*0.94*
RQC	10	0.07	*0.03*	0.21	0.22	2.08	0.27
	12	0.24	*0.08*	0.17	0.31	16.07	0.57
	14	680.70	*0.18*	0.20	0.40	60.09	0.91
	16	2153.49	0.26	*0.17*	0.44	178.73	1.62

- **quimb**[4]: Simulation tool developed for cotengra that uses matrices to represent the information.
- **DDSIM**[5]: Simulation tool that uses QMDDs to represent the information.

The results are presented in Table 5. As can be observed, the DDSIM tool yields the best results by far in all cases except with the RQC circuits, where the fastest tools are FTDD or quimb using the ordering method provided by cotengra. Clearly, DDSIM is using a very efficient implementation of the QMDD diagrams and of the iterative ordering method for very different types of circuits. The second fastest tool is FTDD, which also uses DDs to store and manipulate the circuits. Results also show that matrix-based tools such as quimb and GTN have many problems to store the information and can only simulate circuits with very few qubits, especially if they are circuits with many gates, such as the QWalk circuits. All in all, the results show the advantage of employing DDs in order to reduce the spatial and temporal cost of the simulations and to be able to carry out simulations of larger circuits.

5.2.4 Comparison between circuit implementations

The cost of contraction depends not only on the quantum algorithm we are solving, but also on how it is implemented as a quantum circuit. The same algorithm can have very different implementations depending on the gate set used or the optimizations carried out by the compiler. When analyzing experimental

[4] https://quimb.readthedocs.io

[5] https://github.com/cda-tum/mqt-ddsim

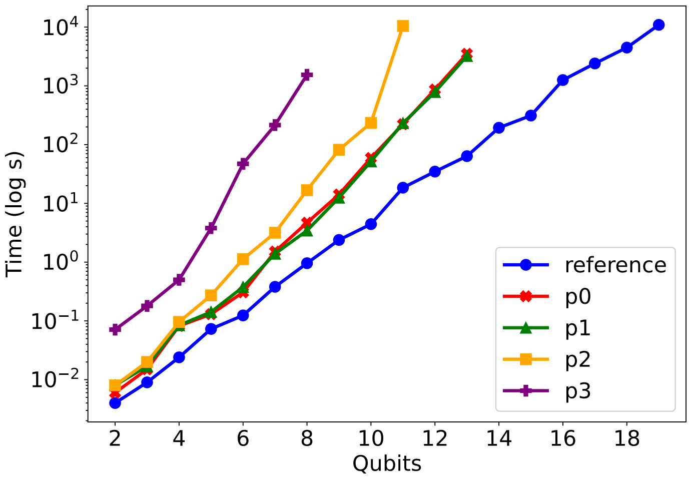


Fig. 3 Comparison of the unitary simulation time of the reference `QFT` circuit versus four compiled variants using the `iterative` method in the FTDD tool

results, it can be misleading to rely only on the name of the algorithm solved (QFT, Grover, QPE, etc.). Various implementations of these algorithms can result in contraction costs that differ by several orders of magnitude. This section compares the contraction costs of different circuits implementing the same algorithm: `QFT`. A circuit of this type, obtained from [51], serves as a reference. From this base, four other circuits were generated, named `qft_p0`, `qft_p1`, `qft_p2`, and `qft_p3`, each using a different set of gates. The four circuits were generated using Qiskit's `transpile` compilation function with the same parameters and optimization level 3. The gate set used for each circuit is detailed as follows:

- qft_p0: id, u1, u2, u3, cx.
- qft_p1: id, rz, sx, x, cx, reset.
- qft_p2: id, rz, sx, x, cz, reset.
- qft_p3: rz, sx, x, ecr.

Figure 3 shows the contraction times of these circuits using the `iterative` algorithm. The figure illustrates the enormous influence that the gates used to implement a circuit can have on the contraction cost, which in the case shown is several orders of magnitude. For instance, the reference circuit, with 8 qubits, has a contraction cost of 0.95 seconds, while the `qft_p3` circuit, with the same number of qubits, has a cost of 1539.17 seconds. The figure illustrates that the size of circuits, range of contracted tensors, and savings achievable through TDDs vary significantly depending on the gates utilized.

6 Conclusions and future work

In this work, we have implemented and evaluated two methods based on DDs to order the contractions of tensor networks using the FTDD tool: `iterative` and `blocks` methods. TDDs combine the advantages of tensor networks with the use of DDs to reduce the time and spatial cost of contraction. The `iterative` method reduces the average size of the intermediate diagrams produced during contraction, thereby decreasing the spatial and temporal cost of the process. The `blocks` method detects and exploits the iterative structure of many quantum circuits to avoid repeating a large number of costly contractions. Experiments on a variety of quantum circuits show that in many cases the proposed methods improve the temporal and spatial performance of other well-known methods such as `sequential` or `cot` orders. In particular, the `blocks` method achieves significant cost reductions for circuits such as those that implement quantum walks or the Grover algorithm.

Finally, the results indicate that the proposed methods utilizing the FTDD tool can reduce the cost of building a representation of the full functionality of a quantum circuit when compared to well-known tools such as quimb that rely on matrix operations. Moreover, the comparison of different simulation tools indicates that DD-based tools, such as FTDD or DDSIM, reduce the simulation time of well-known tools using matrices, such as quimb and GTN. Besides, using DDs much larger circuits can be simulated.

As a future goal, we aim to develop methods to simulate efficiently quantum circuits on classical computers using TDDs. One way to achieve this is by applying parallelization techniques at different levels of quantum circuit simulation. These levels include parallelizing a single contraction between two tensors, parallelizing contractions of different groups of tensors, and applying slicing to the tensor network and parallelizing the contraction of the resulting sub-tensor networks. By combining TDDs with these parallelizing strategies, we intend to reduce the spatial cost of contracting the tensor network and simulate larger circuits.

Author contributions V.L.O. implemented the codes and executed all the experiments on the computing platforms. J.M.B. and M.C. supervised the work and helped to conceive the codes and experiments. All authors discussed the results and contributed to the final version of the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been funded by the project UJI-B2021-26 of the Universitat Jaume I and by grant PID2020-113656RB-C21 funded by MCIN/AEI/10.13039/50110011033 and MICINN.

Data availability Not yet available.

Code availability Not yet available.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication All authors read and approved the final manuscript.

References

1. Yuan X (2020) A quantum-computing advantage for chemistry. Science 369(6507):1054–1055
2. Gerbert P, Rueß F (2018) The next decade in quantum computing and how to play. Boston Consulting Group
3. Sutor RS (2019) Dancing with qubits: how quantum computing works and how it can change the world. Packt Publishing Ltd, Birmingham
4. Forum WE (2022) State of quantum computing: building a quantum economy. https://www3.weforum.org/docs/WEF_State_of_Quantum_Computing_2022.pdf
5. Preskill J (2018) Quantum computing in the NISQ era and beyond. Quantum 2:79
6. Huang C, Zhang F, Newman M, Ni X, Ding D, Cai J, Gao X, Wang T, Wu F, Zhang G et al (2021) Efficient parallelization of tensor network contraction for simulating quantum computation. Nat Comp Sci 1(9):578–587
7. Gray J, Kourtis S (2021) Hyper-optimized tensor network contraction. Quantum 5:410
8. Wu XC, Di S, Dasgupta EM, Cappello F, Finkel H, Alexeev Y, Chong FT (2019) Full-state quantum circuit simulation by using data compression. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp 1–24
9. Bernstein E, Vazirani U (1993) Quantum complexity theory. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing, pp 11–20
10. Vincent T, O'Riordan LJ, Andrenkov M, Brown J, Killoran N, Qi H, Dhand I (2022) Jet:fast quantum circuit simulations with parallel task-based tensor-network contraction. Quantum 6:709
11. Boixo S, Isakov SV, Smelyanskiy VN, Babbush R, Ding N, Jiang Z, Bremner MJ, Martinis JM, Neven H (2018) Characterizing quantum supremacy in near-term devices. Nat Phys 14(6):595–600
12. Chi-Chung L, Sadayappan P, Wenger R (1997) On optimizing a class of multi-dimensional loops with reduction for parallel execution. Parallel Process Lett 7(02):157–168
13. Akers SB (1978) Binary decision diagrams. IEEE Trans comput 27(06):509–516
14. Zulehner A, Wille R (2018) Advanced simulation of quantum computations. IEEE Trans Comput Aided Des Integr Circuits Syst 38(5):848–859
15. Hong X, Zhou X, Li S, Feng Y, Ying M (2022) A tensor network based decision diagram for representation of quantum circuits. ACM Trans Des Autom Electr Syst (TODAES) 27(6):1–30
16. Zhang Q, Saligane M, Kim HS, Blaauw D, Tzimpragos G, Sylvester D (2024) Quantum circuit simulation with fast tensor decision diagram. arXiv preprint arXiv:2401.11362
17. Quantiki (2023) List of QC simulators. https://quantiki.org/wiki/list-qc-simulators
18. Dahi Z, Alba E, Gil-Merino R, Chicano F, Luque G (2023) A survey on quantum computer simulators. https://api.semanticscholar.org/CorpusID:259257304
19. Jones T, Brown A, Bush I, Benjamin SC (2019) QuEST and high performance simulation of quantum computers. Sci Rep 9(1):1–11

20. Guerreschi GG, Hogaboam J, Baruffa F, Sawaya NP (2020) Intel quantum simulator:a cloud-ready high-performance simulator of quantum circuits. Quantum Sci Technol 5(3):034007
21. Markov IL, Shi Y (2008) Simulating quantum computation by contracting tensor networks. SIAM J Comp 38(3):963–981
22. Villalonga B, Boixo S, Nelson B, Henze C, Rieffel E, Biswas R, Mandrà S (2019) A flexible high-performance simulator for verifying and benchmarking quantum circuits implemented on real hardware. npj Quantum Inform 5(1):86
23. Brennan J, Allalen M, Brayford D, Hanley K, Iapichino L, O'Riordan LJ, Doyle M, Moran N (2021) Tensor network circuit simulation at exascale. In: 2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS), IEEE, pp 20–26
24. Vincent T, O'Riordan LJ, Andrenkov M, Brown J, Killoran N, Qi H, Dhand I (2022) Jet: fast quantum circuit simulations with parallel task-based tensor-network contraction. Quantum 6:709
25. Lykov D, Schutski R, Galda A, Vinokur V, Alexeev Y (2022) Tensor network quantum simulator with step-dependent parallelization. In: 2022 IEEE International Conference on Quantum Computing and Engineering (QCE), IEEE, pp 582–593
26. Zhang S-X, Allcock J, Wan Z-Q, Liu S, Sun J, Yu H, Yang X-H, Qiu J, Ye Z, Chen Y-Q et al (2023) Tensorcircuit: a quantum software framework for the NISQ era. Quantum 7:912
27. Huang C, Zhang F, Newman M, Cai J, Gao X, Tian Z, Wu J, Xu H, Yu H, Yuan B, et al (2020) Classical simulation of quantum supremacy circuits. arXiv preprint arXiv:2005.06787
28. Gray J (2018) quimb: a python package for quantum information and many-body calculations. J Open Sour Softw 3(29):819
29. Gogate V, Dechter R (2012) A complete anytime algorithm for treewidth. arXiv preprint arXiv:1207.4109
30. Strasser B (2017) Computing tree decompositions with flowcutter: PACE 2017 submission. arXiv preprint arXiv:1709.08949
31. Gray J (2023) Cotengra. https://github.com/jcmgray/cotengra
32. Schlag S, Heuer T, Gottesbüren L, Akhremtsev Y, Schulz C, Sanders P (2023) High-quality hypergraph partitioning. ACM J Exp Algorithm 27:1–39
33. Daniel G, Gray J et al (2018) opt_einsum. A python package for optimizing contraction order for einsum-like expressions. J Open Sour Softw 3(26):753
34. Lee C-Y (1959) Representation of switching circuits by binary-decision programs. Bell Syst Tech J 38(4):985–999
35. Bryant RE (1986) Graph-based algorithms for boolean function manipulation. Comput IEEE Trans 100(8):677–691
36. Bergman D, Cire AA, Van Hoeve W-J, Hooker J (2016) Decision diagrams for optimization, vol 1. Springer, Cham
37. Oliva VL (2022) Fundamentos de la computación cuántica. TEMat 6:31–47
38. Dirac PAM (1939) A new notation for quantum mechanics. Math Proceed Camb Philos Soc 35(3):416–418
39. Bridgeman JC, Chubb CT (2017) Hand-waving and interpretive dance: an introductory course on tensor networks. J phys A Math Theor 50(22):223001
40. Miller DM, Thornton MA (2006) QMDD: a decision diagram structure for reversible and quantum circuits. In: 36th International Symposium on Multiple-Valued Logic (ISMVL'06), IEEE, pp 30–30
41. Niemann P, Wille R, Miller DM, Thornton MA, Drechsler R (2015) QMDDs: efficient quantum function representation and manipulation. IEEE Trans Comput Aided Des Integr Circuits Syst 35(1):86–99
42. Burgholzer L, Raymond R, Sengupta I, Wille R (2021) Efficient construction of functional representations for quantum algorithms. In: Reversible Computation: 13th International Conference, RC 2021, Virtual Event, Springer, 7–8 July 2021, Proceedings, pp 227–241
43. Grover LK (1996) A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, pp 212–219
44. Venegas-Andraca SE (2012) Quantum walks: a comprehensive review. Quantum Inform Process 11(5):1015–1106
45. Brassard G, Hoyer P, Mosca M, Tapp A (2002) Quantum amplitude amplification and estimation. Contemp Math 305:53–74
46. Nielsen MA, Chuang IL (2011) Quantum computation and quantum information: 10th anniversary edition. Cambridge University Press, New York, USA

47. Cross AW, Bishop LS, Smolin JA, Gambetta JM (2017) Open quantum assembly language. arXiv preprint arXiv:1707.03429
48. Efthymiou S, Ramos-Calderer S, Bravo-Prieto C, Pérez-Salinas A, García-Martín D, Garcia-Saez A, Latorre JI, Carrazza S (2021) Qibo: a framework for quantum simulation with hardware acceleration. Quantum Sci Technol 7(1):015018
49. Mei J, Bonsangue M, Laarman A (2024) Simulating quantum circuits by model counting. In: International Conference on Computer Aided Verification, Springer, pp 555–578
50. Niemann P, Wille R, Drechsler R (2013) On the "Q" in QMDDs: efficient representation of quantum functionality in the QMDD data-structure. In: Reversible Computation: 5th International Conference, RC 2013, Victoria, BC, Canada, 4-5 July 2013. Proceedings 5, Springer, pp 125–140
51. Quetschlich N, Burgholzer L, Wille R (2022) MQT bench: benchmarking software and design automation tools for quantum computing. arXiv. MQT Bench is available at https://www.cda.cit.tum.de/mqtbench/
52. Choi J, Kim J (2019) A tutorial on quantum approximate optimization algorithm (QAOA): Fundamentals and applications. In: 2019 International Conference on Information and Communication Technology Convergence (ICTC), IEEE, pp 138–142
53. Fedorov DA, Peng B, Govind N, Alexeev Y (2022) VQE method: a short survey and recent developments. Mater Theor 6(1):2
54. Wang P, Wang D, Ji Y, Xie X, Song H, Liu X, Lyu Y, Xie Y (2019) QGAN: quantized generative adversarial networks. arXiv preprint arXiv:1901.08263
55. Narayanan A, Menneer T (2000) Quantum artificial neural network architectures and components. Inform Sci 128(3–4):231–255
56. Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, Biswas R, Boixo S, Brandao FG, Buell DA et al (2019) Quantum supremacy using a programmable superconducting processor. Nature 574(7779):505–510