Adaptive algorithm for quantum circuit simulation

Roman Schutski,[1] Danil Lykov,[2] and Ivan Oseledets[1]
[1]*Center for Computational and Data-Intensive Science and Engineering, Skoltech, Skolkovo Innovation Center, Moscow Region 121205, Russian Federation*
[2]*Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region 141701, Russian Federation*

(Received 4 July 2019; accepted 19 March 2020; published 29 April 2020)

Efficient simulation of quantum computers is essential for the development and validation of near-term quantum devices and the research on quantum algorithms. Up to date, two main approaches to simulation have been use, based on either full-state or single-amplitude evaluation. We propose an algorithm that efficiently interpolates between these two possibilities. Our approach elucidates the connection between quantum circuit simulation and partial evaluation of expressions in tensor algebra.

DOI: 10.1103/PhysRevA.101.042335

I. INTRODUCTION

An intense interest in quantum computing in recent years has led to the increase of size and capabilities of experimental quantum computers. Promising physical realizations of quantum computing devices have been proposed in recent years [1,2], which bolsters the expectations that a long-desired quantum supremacy will be reached [3–5].

In the meantime, substantial progress has been made in understanding quantum computation and developing classical simulators of quantum circuits. Efficient simulators have been developed for highly parallel computers, such as Sunway Taihulight [6]. At the moment the simulation software is aimed at either one of two tasks. The first one is predicting the probability of measuring a particular binary string as the result of a quantum program, or single-amplitude simulation. The second is obtaining the full distribution of quantum circuit outputs, or full-state simulation. The first approach was found to be more economical in terms of memory a classical computer has to use, thus allowing the simulation of few amplitudes of larger quantum circuits on up to 100 qubits [7]. On the other side, the second approach may be preferred when the full-state information is needed, such as in Shor's algorithm [8].

In this paper, we present a unified approach to quantum circuit simulation. The user can choose the number of probabilities of bitstrings to simulate in a single pass. Our algorithm allows one to compromise between the amount of available computational resources and the overall time of the simulation. We build our work on the connection of graphical models and quantum circuits introduced by Markov and Shi [9] and later developments by Boixo *et al.* [10] and other authors. Here, we concentrate on an inherently sequential algorithm and defer the discussion of parallelization strategies to a subsequent publication. However, an interested reader is referred to Refs. [6,7,11] for efficient parallel simulation algorithms. During the development of this paper an interesting work was presented by Pednault *et al.* [11]. We find that our approach is more straightforward, as it disentangles the problem of multiple-amplitude simulation from the parallelization. We defer a more detailed comparison to a later section. An overview of the paper is as follows.

In Sec. III, we review the connection of quantum circuits, tensor diagrams, and statistical graphical models. We then proceed by describing a basic algorithm for circuit simulation based on Refs. [9,10]. In Sec. IV we formulate the main problem solved in this work, e.g., batch simulation of amplitudes. To solve it, we recall the tree decomposition of graphs and its connection to the problem of ordering of graph nodes. We then propose an algorithm to transform graph orderings while preserving treewidth (e.g., the quality) of the given ordering. To achieve a proper transformation we use the connection of tree decomposition and chordal graphs, as explained in Secs. IV B and IV C. Numerical experiments are listed in Sec. IV D. Finally, we conclude in Sec. V with final remarks and outline possible future research. We present a short overview of the literature on quantum circuit simulation in Sec. II, we include another technique in the Appendix for comparison with our approach.

II. RELATED WORK

Here we put our method in the scope of existing research. We suggest that the reader interested in implementation move on to Sec. III and revisit this section later.

The problem of efficient tensor contraction has been approached multiple times in the field of many-body physics and quantum computing. Some older works are based on the sequential application of sparse matrices to the state vector, such as in Ref. [12]. The authors issued a follow-up paper recently [13]. Their simulator can evaluate both full sets and subsets of amplitude tensor. This direct simulation procedure, however, requires a lot of nontrivial techniques to make it efficient, especially if parallel operation is considered. Another problem is that it is hard to analyze the effectiveness of the algorithm compared to theoretical bounds on the numerical cost [14]. The latter fact has lead to the previously believed margin of 50 qubits for "quantum supremacy."

The seminal work of Markov and Shi [9] introduced tensor networks for quantum algorithm simulations and showed

that treewidth is a natural measure of simulation hardness. The graph-based notation became standard in tensor network literature a decade ago [15]. Following Markov and Shi, several groups developed highly efficient algorithms for quantum circuit simulation based on this representation (see Refs. [6,11,16,17] for more details). The previous margin of 50 qubits was lifted, as has been demonstrated by multiple authors [6,11,17]. Usually, these simulators are capable of evaluating full-state vectors as well as some subsets of the amplitudes. A similar program was created for contraction of tensors emerging in the many-body physics community [16]. The drawback of the approaches based on tensor diagrams is the hardness of the development of efficient codes and the theoretical performance analysis, especially if parallelization is involved. To see why, let us note that classical tensor networks were developed to represent pairwise contractions. Quantum circuits often involve multiple diagonal gates, which allows for significant computational savings. The treewidth of classical diagram's graphs is higher than optimal (see the Appendix in Ref. [10]). Traditional network notation can be understood as a hypergraph to eliminate this drawback, as was done in Ref. [11]. However, the theory of hypergraphs is less known to the general scientific community.

Recently Boixo *et al.* [10] proposed to consider line graphs of the classical tensor networks, which has multiple benefits. First, it establishes the connection of quantum circuits with probabilistic graphical models, allowing for knowledge transfer between the fields. Second, these graphical models avoid the overhead of traditional diagrams for diagonal tensors. Moreover, the treewidth is a universal measure of complexity for these models and links the complexity of quantum states to the well-studied problems in graph theory, a topic we hope to explore in future works. Additionally, simple parallelization of the simulator is possible, as demonstrated in the work of Chen *et al.* [7] The only disadvantage of the line graph approach is that it has limited usability to simulate subtensors of amplitudes, which we are going to fulfill in this article.

Last, we have to mention that multiple approximate methods are currently being developed for circuit simulation. Very recent work of Carrasquilla and coworkers [18] presents a neural-network-based approach. Pan and co-workers [19] devised an approximate algorithm based on tensor network transformation. The extension of our approach with approximation techniques may be a perspective direction of research.

III. QUANTUM CIRCUIT SIMULATION ALGORITHM

In this section we describe a procedure for efficient quantum circuit evaluation. We first set up the notation and then review the current state-of-the-art method for numerical simulation of quantum circuits.

A. Tensor networks and graphical models

A quantum program describes an evolution of the initial state $|0\rangle$ of a system of n qubits. Any evolution of a physical system corresponds to a unitary operator. Thus, the result of a quantum circuit is a state $|\psi\rangle$, which is a linear transformation of the input state: $|\psi\rangle = \mathcal{U}|0\rangle$. Usually, the transformation U is performed in several steps corresponding to clock cycles of a quantum computer. Suppose the transformation is described by a depth d circuit. We introduce the following notation:

$$\begin{aligned} \mathcal{U}\,|0\rangle &= \mathcal{U}^d \ldots \mathcal{U}^2\mathcal{U}^1\,|0\rangle\,, \\ |s^{t+1}\rangle &= \mathcal{U}^t\,|s^t\rangle\,, \quad |s^0\rangle = |0\rangle\,. \end{aligned} \tag{1}$$

Here U_t are unitary matrices acting at the tth clock cycle and $|s^t\rangle$ is the state vector. In the simplest case the initial state is taken to be a product of single-qubit states $|0\rangle = |0_0\rangle \otimes \cdots \otimes |0_n\rangle$. A naive simulation algorithm would take the initial vector $|0\rangle$ and apply the matrices $\mathcal{U}^t$ to it. This procedure lies behind full-state circuit simulation. To calculate an amplitude of a bitstring x, one would evaluate a dot product $\langle x|s^d\rangle$:

$$\sigma(x) = \langle x|s^d\rangle = \sum_{i=1}^{n} \langle x_i|s^d\rangle\,. \tag{2}$$

The probability of x is then the modulus squared of the amplitude. Note, however, that it is hard to perform full-state simulation efficiently. A naive algorithm would need to operate on vectors of size 2^n. Also, the matrices $\mathcal{U}^t$ are highly sparse, at least if they represent transformations achievable with single- and two-qubit gates in modern experimental hardware. Here and later in the paper, we chose to work with the following universal set of one- and two-qubit gates: $\{X^{1/2}, Y^{1/2}, cZ, T, H\}$; the same reasoning, however, applies to any quantum gate.

An alternative to full-state simulation would be the evaluation of one or several amplitudes from Eq. (2) without explicitly forming $|s^d\rangle$. The latter approach provides several benefits. First of all, we can avoid storing the high-dimensional state vector $|s^d\rangle$ in computer memory. Second, it may be easier to use the internal structure of the operators $\mathcal{U}^t$ to perform calculations efficiently. Let us introduce a set of variables to denote the state at different cycles of the circuit:

$$\{s\}_i^t, \quad s \in [0, 1], \quad i \in [1, n], \quad t \in [0, d]. \tag{3}$$

We slightly abuse notation here, as $|s_i^t\rangle$ denotes a state of the ith qubit at the tth cycle, and s_i^t is a binary variable indexing this state. The same notation is used for the initial and final states, e.g., $|s_i^0\rangle = |0_i\rangle$ and $|s_i^d\rangle = |x_i\rangle$. Consider the circuit shown in Fig. 1.

We start with a product state $|0\rangle$ on the right. As the program proceeds, the states of individual qubits are changed by gate application. Note that the gates T and cZ do not change the basis of the single-qubit subspaces they act on (they only multiply basis vectors by constants), and hence $|s_i^t\rangle = |s_i^{t+1}\rangle$ for those qubits. In contrast, the nondiagonal gates $\{X^{1/2}, Y^{1/2}, H\}$ mix basis vectors of the appropriate qubit subspaces, and new variables $|s_i^{t+1}\rangle$ have to be introduced for the resulting bases. In Fig. 1 only unique variables are shown. The expression for the single amplitude in Eq. (2) can be rewritten as

$$\begin{aligned} \sigma(x) &= \langle x|\mathcal{U}|0\rangle \\ &= \sum_{\{s_i^t\}} \left\langle x_i\middle|\mathcal{G}_i^d\middle|s_i^{d-1}\right\rangle \cdots \left\langle s_i^{t+1}s_j^{t+1}\middle|\mathcal{G}_{ij}^t\middle|s_i^t s_j^t\right\rangle \cdots \left\langle s_i^1\middle|\mathcal{G}_i^1\middle|0_i\right\rangle, \\ \mathcal{G}_i^t &\in \{X^{1/2}, Y^{1/2}, T, H\}, \qquad \mathcal{G}_{ij}^t = cZ. \end{aligned} \tag{4}$$

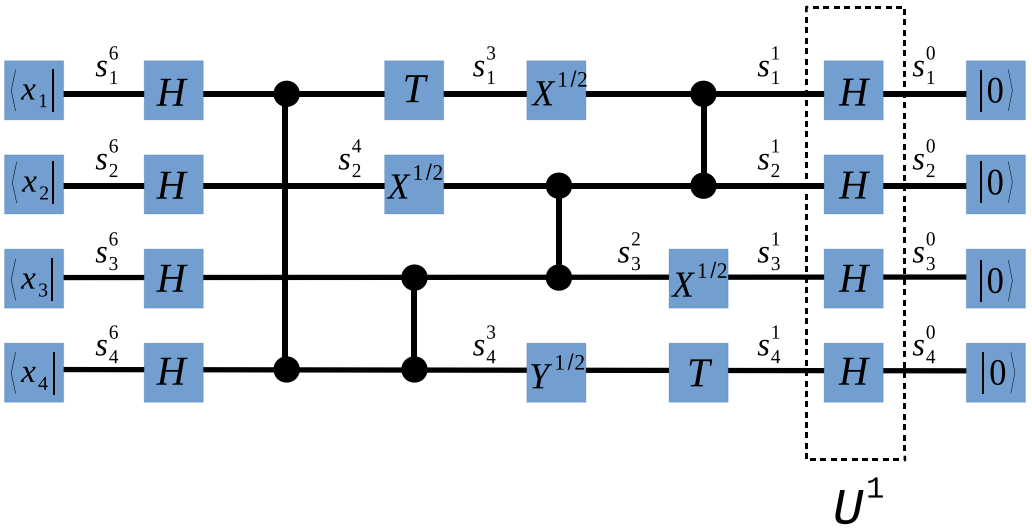

FIG. 1. Example of a quantum circuit drawn as a tensor network. The state of the ith qubit at the tth clock cycle is denoted by $\{s\}_i^t$. (Only unique states are shown; e.g., $s_1^2 = s_1^1$ is omitted.)

Equation (4) can be interpreted as a discrete Feynman path integral. On the other hand, one can easily see that the evaluation of the amplitude $\sigma(x)$ in Eq. (4) is equal to the contraction of the tensor network shown in Fig. 1. (For the introduction to the graphical notation used for tensor networks, please refer to Ref. [20].) It is well known, however, that the numerical cost of tensor contractions dramatically depends on the order of operations. Following Markov and Shi [9], let us introduce another type of graphical model to denote quantum circuits, which is better suited for the estimation of numerical costs.

In traditional notation, a tensor network is represented by a graph with nodes standing for tensors and edges denoting their indices. In our notation, we use nodes to denote unique indices, and tensors are denoted by cliques (fully connected subgraphs). Note that tensors, which are diagonal along some of the axes and hence can be indexed with fewer variables, are depicted by cliques of size lower than the dimension of the corresponding tensor. For a special case of vectors or diagonal matrices, self-loop edges are used. Figure 2 lists the notation for the gates used in this work.

A graphical model, which is equivalent to the circuit in Fig. 1, is shown in Fig. 3 (self-loops are omitted for simplicity). As was pointed out by Boixo *et al.* [10], this representation of tensor contractions is traditional in Bayesian network literature. Notice that provided a quantum circuit in a traditional form, one can easily build its probabilistic model representation. To do that, one has to replace all edges carrying nonequivalent single-qubit states with nodes, and all gates with cliques. The diagonal structure of cZ gate tensors leads to significant simplification of the resulting graphs.

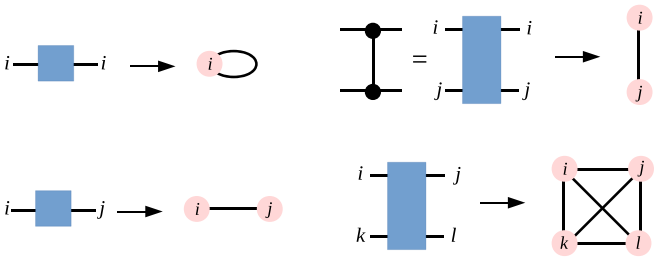

FIG. 2. The mapping between two graph-based notations of tensor networks.

B. Simulation of quantum circuits

Having set up the notation, let us proceed with a description of a basic procedure for the evaluation of tensor networks. This algorithm was developed in the context of probabilistic models under the name of bucket elimination [21] or the variable elimination algorithms [22].

As an example, let us consider the contraction of a simple tensor network:

$$\sum_{ijklmn} A_{ij}B_{jk}C_{ikl}D_{km}E_{ln}F_{mn} = \sigma. \tag{5}$$

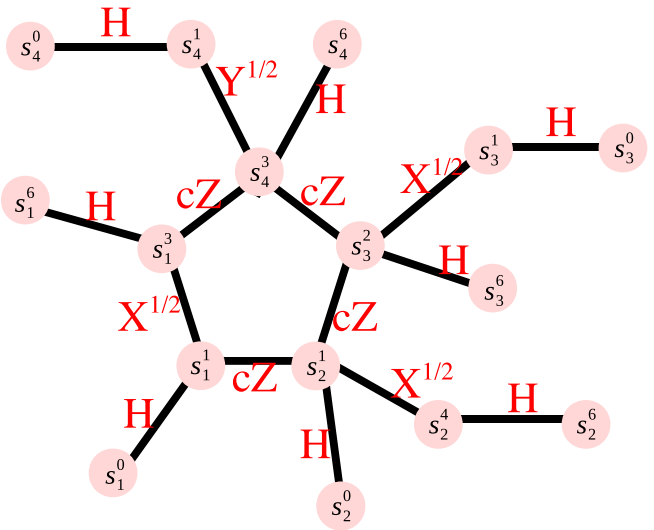

FIG. 3. Alternative representation (graphical model) of the circuit in Fig. 1. Gate tensors are shown in red (light gray); self-loops are omitted.

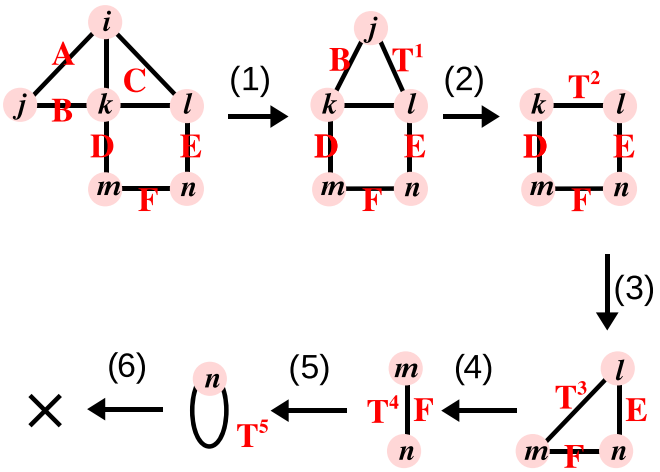

FIG. 4. Contraction of a tensor network from Eq. (5) in graphical form. The sequence of contractions π is the same as that in Eq. (6). Labels of tensors are shown in red (gray).

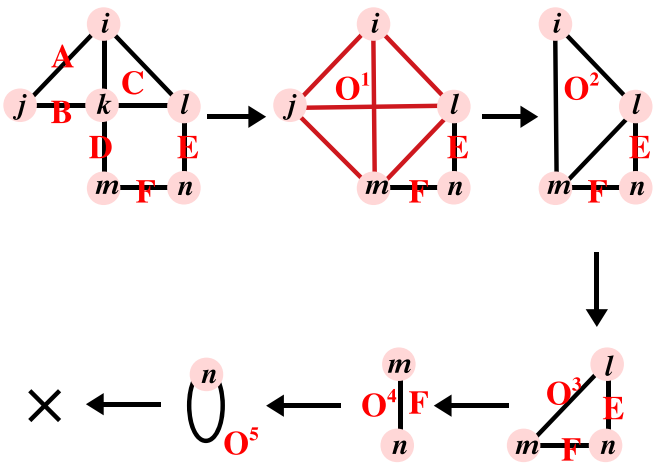

FIG. 5. Alternative contraction of the tensor network in Eq. (5). The maximal clique of size 4 is highlighted in red (gray). This sequence of contractions is not optimal.

The graphical model of this network is shown in Fig. 4. We choose the order of indices as $\pi = \binom{i\ j\ k\ l\ m\ n}{1\ 2\ 3\ 4\ 5\ 6}$; e.g., i is first, j is second, etc. In the bucket elimination procedure the indices are contracted one at a time in order fixed by π, until no indices are left. The sequence of the expressions evaluated in the algorithm is listed below. Assuming the dimension of all indices is L, we also list numerical costs of the operations:

$$\begin{aligned}
&(1)\ \sum_i A_{ij}C_{ikl} = T^1_{jkl} \quad O(L^4),\\
&(2)\ \sum_j B_{jk}T^1_{jkl} = T^2_{kl} \quad O(L^3),\\
&(3)\ \sum_k D_{km}T^2_{kl} = T^3_{ml} \quad O(L^3),\\
&(4)\ \sum_l E_{ln}T^3_{ml} = T^4_{nm} \quad O(L^3),\\
&(5)\ \sum_m T^4_{nm} = T^5_n \quad O(L^2),\\
&(6)\ \sum_n T^5_n = \sigma \quad O(L).
\end{aligned} \tag{6}$$

The sequence of transformations of the graphical model corresponding to Eq. (6) is shown in Fig. 4: (2)–(6).

At each step, the contracted variable is removed from the graph, and all its neighbors form a clique. This clique corresponds to the next intermediate in the sequence. Note that the order of the cliques formed at each step corresponds to the exponent of the scaling of numerical cost.

The computational cost of the tensor network contraction is highly dependent on the order of operations. To illustrate this let us consider an alternative order $\tilde{\pi} = \binom{k\ j\ i\ l\ m\ n}{1\ 2\ 3\ 4\ 5\ 6}$ for evaluating Eq. (5). The corresponding sequence of graphical models is shown in Fig. 5. Note that the size of the maximal clique corresponding to order $\tilde{\pi}$ is 4, which translates to the intermediate of order 4 and the overall scaling $O(L^5)$ of the numerical effort.

Finding the elimination order of a graph is equivalent to the calculation of its tree decomposition; the size of the maximum clique of an order π is treewidth + 1. Tree decomposition is NP-hard for general graphs [23], and a similar hardness result is known for the optimal tensor contraction problem [24]. However, several exact and approximate algorithms for tree decomposition have been developed in graph theory literature; for references, please see Refs. [23,25–28]. For our simulations, we used an exact algorithm of Gogate and Rechter [25]. Having reviewed the procedure for calculation of a single amplitude, let us consider the case of multiple amplitudes, which is the main topic of this article.

IV. BATCH CIRCUIT SIMULATION

A. Simulation of multiple amplitudes

The procedure we used to calculate a single amplitude can be easily extended to calculate any subtensor of the full-amplitude tensor. Suppose we are interested in amplitudes of two bitstrings differing only in the value of the first qubit, e.g., $x^0 = (0, s^d_2, \ldots, s^d_n)$ and $x^1 = (1, s^d_2, \ldots, s^d_n)$. Let us note that the expressions for the amplitudes of σ^0 and σ^1 differ only by the values of the state vectors of the first qubit, which are $|0\rangle$ and $|1\rangle$, respectively. One could merge both expressions and introduce an additional variable s^{d+1}_1 to index the result $\sigma(s^{d+1}_1)$ (which is a vector of size 2). The same procedure can be implemented for any combination of output qubits; thus, any subtensor of the full-amplitude tensor can be encoded. A graphical representation of the extended amplitude expression is shown in Fig. 6. We have to mention that the same procedure can be used to evaluate not only the probabilities of multiple output states but also the evolution of multiple input states. This approach can be used to simulate the dynamics of mixed states, although we do not elaborate on this in the current article.

In order to evaluate multiple amplitudes, the resulting extended expressions have to be contracted only partially (the indices of the amplitude subtensors should not be summed over). Partial contraction can be achieved by stopping the

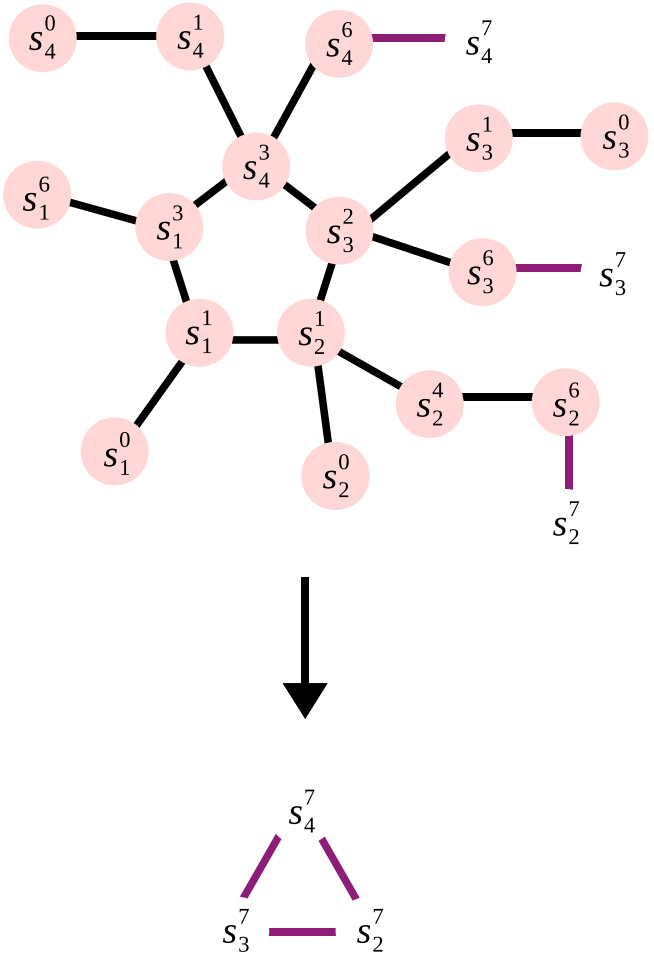

FIG. 6. Evaluation of amplitude subsets. Left: Extended amplitude expression to evaluate all amplitudes of qubits 2, 3, and 4. Right: Resulting amplitude tensor.

bucket elimination algorithm [21,22] when all necessary indices are eliminated and (possibly) merging the final set of tensors. Notice that the result of the evaluation of all amplitudes for c qubits will result in a tensor with 2^c elements, which will be mirrored by a clique (a fully connected subgraph) with c nodes in our graphical notation (Fig. 6, right).

After selecting a subset of nodes to leave in the result, one still faces the problem of choosing an optimal order of variable elimination to implement partial contractions. Let us turn to the discussion of a possible solution.

B. Node ordering and chordal graphs

To properly introduce the procedure of finding elimination orders for partial contractions, let us first highlight the connection of elimination orders and chordal graphs. Chordal graphs (also called triangulated graphs) are the ones that do not have cycles of length higher than 3. Many problems which are hard on general graphs can be solved on chordal graphs in polynomial time (for example, the maximum clique problem [29]). An extensive introduction to the properties of chordal graphs and related algorithms can be found in Ref. [30].

Algorithm 1. Building chordal graph from the elimination order

Input: $G = (V, E), \pi : V \to \mathcal{N}, \quad \pi = \{(v_i, i)\}_{i=1}^{|V|}$
Output: $H = (V, \tilde{E}), \quad H$ is chordal

```
function BUILD_CHORDAL_GRAPH G, π
    Ẽ ← E
    for i ∈ [1, ..., |V|] do
        v ← π^{-1}(i)
        U = ∅
        for w in neighbors (v) do
            if π(w) > i then
                U ← U ∪ w
            end if
        end for
        for x, y in pairs (U) do
            Ẽ ← Ẽ ∪ (x, y)
        end for
    end for
end function
```

We employ chordal graphs because of their relation to node orderings. Consider the bucket elimination procedure described before, but without node removal. Specifically, given a graph G and a node order π, one would loop over the nodes according to π and for each node connect all of its neighbors who have higher order in π. It can be shown [30] that this procedure will always produce a chordal graph. Indeed, if the initial graph has any cycle with four or more nodes, connecting the neighbors of any node in the cycle will introduce a chord, thus breaking a cycle into smaller, three-node cycles. The resulting chordal graph is also called a fill-in graph in this context.

A formal algorithm for building a fill-in graph H given an initial graph G and an elimination order π is listed in Algorithm 1. A corresponding graphical representation of the algorithm is provided in Fig. 7. An important remark has to

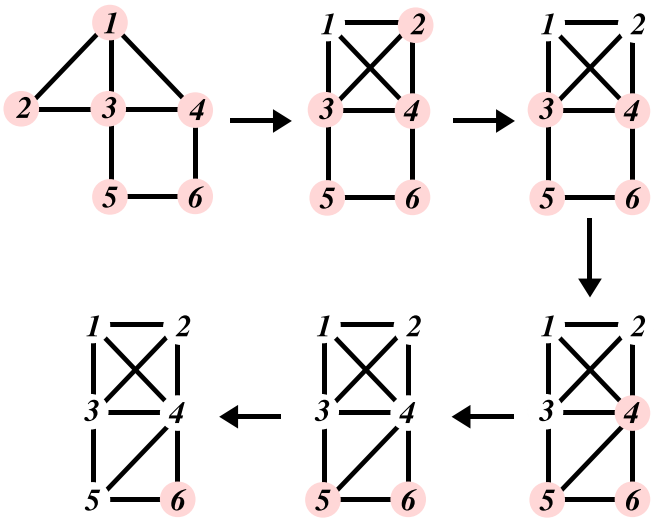

FIG. 7. Building a chordal graph from the elimination order. The graph corresponds to the tensor network in Eq. (5). The nodes are labeled according to their order.

be made here. Any elimination order π will produce a chordal graph, but this does not imply that there is a one-to-one correspondence. Multiple orders can result in the same fill-in graph [31]; we employ this fact in the next section.

The size of the maximum clique in the fill-in graph equals the *treewidth* by construction [26]. The problem of searching the best elimination order for a graph G thus can be formulated in terms of the search of an optimal fill-in graph. Formally, given a graph $G = (V, E)$, the task of finding an elimination order π with minimal treewidth is equivalent to finding a chordal graph $H = (V, \tilde{E})$, $\tilde{E} \in E$, such that the size of its maximum clique is minimized.

Provided a chordal graph H is found, any of its elimination orders that do not introduce additional edges, and hence do not change the graph H, will have the same treewidth. Chordal graphs thus provide the means of building equivalent (in terms of treewidth) elimination orders. In contrast with arbitrary graphs, finding elimination orders of chordal graphs can be done in linear time [31]. We now turn to the description of the procedure of the building of equivalent elimination orders of chordal graphs.

C. Finding restricted elimination orders

Let us now find an optimal elimination order for multiple-amplitude evaluation, as described in Sec. IV A. In essence, we want to find an order with minimal treewidth, such that some set of nodes will be at the end of this order. Putting it formally, for a graph $G = (V, E)$ and a subset of nodes $C \in V$, we want to find an order π with minimal treewidth, such that for any nodes $v \in C$ and $w \in V \setminus C$ the order of v is higher than the order of w: $\pi(v) > \pi(w)$.

Our idea is to calculate an optimal unrestricted elimination order $\tilde{\pi}$ (not necessarily having C at the end) and then to use the connection between the elimination orders and the chordal graphs to transform it to the desired order π. Essentially, we employ the result of Bodlaender *et al.* [26] to devise a procedure for building π. Our approach is outlined below.

(i) Check if C induces a clique in G. If $G[C]$ is not a clique, turn $G[C]$ into a fully connected subgraph. This step ensures that the condition stated in Ref. [26], Lemma 10, is satisfied: If C is a clique, then there always exists an elimination order with C at the end. A graph $\tilde{G}$ is produced as a result of (possibly) turning C into a clique.

(ii) Find an elimination order $\tilde{\pi}$ of $\tilde{G}$ using an exact (NP-hard) or a heuristic algorithm. We use the branch and bound algorithm of Gogate and Dechter [25] with the time limit of 60 s (to obtain an exact solution the algorithm has to be given a very long time).

(iii) Build a chordal graph H using Algorithm 1.

(iv) Provided with a set C and a chordal graph H, construct a new order π with the help of the restricted maximum cardinality search (MCS) algorithm. The order π has the same treewidth as the order $\tilde{\pi}$, and the nodes in C are placed at the end in π.

Essentially, in our approach, we transform an arbitrary solution to the tree decomposition problem to the one that satisfies our restrictions (places all nodes in C to the end) and has the same quality (same treewidth). The last ingredient to complete the procedure is the restricted cardinality search algorithm. We modified the original algorithm from Ref. [31]. The resulting pseudocode is provided in Algorithm 2.

Algorithm 2. Computing an elimination order with a set of nodes C at the end

Input: $H = (V, E)$, H is chordal, $C \in V$, C is clique
Output: $\pi : V \to \mathcal{N}$, $\pi = \{(v_i, i)\}_{i=1}^{|V|}$

```
function RESTRICTED-MCS(H, C)
    for v ∈ V do
        cardinality(v) ← 0
    end for
    for i ∈ [|V|, |V| − 1, . . . , 1] do
        if C ≠ ∅ then
            pick v ∈ C, C ← C \ {v}
        else
            pick v ∈ V with maximum cardinality
            V ← V \ {v}
        end if
        π ← π ∪ (v, i)
        for w ∈ neighbors (v), w ∉ π do
            cardinality (w) ← cardinality (w) + 1
        end for
    end for
end function
```

In Algorithm 2, each node v of the graph H is assigned a counter "cardinality," which holds the number of labeled neighbors of v. At each step, we label the next node with maximal cardinality, breaking ties arbitrarily. The order is built in a reversed form. In the beginning, the nodes in C are labeled (to be last), and then the rule stated before is applied. Note that if at step i a node v is selected, then in the next steps all neighbors of v, which belong to the maximal clique K, $v \in K$, will be labeled. Overall, the procedure in Algorithm 2 is polynomial in the number of nodes $|V|$.

Let us now demonstrate the benefits of the proposed approach with numerical examples.

D. Numerical examples

The methods developed in previous sections were used to implement a quantum circuit simulator. As numerical examples we use the simulation of random quantum circuits from

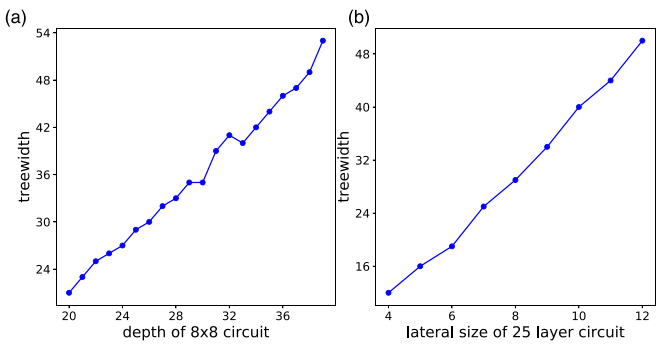

FIG. 8. Treewidth dependence on the size of a random quantum circuit. (a) Dependence of treewidth on the depth of a random circuit. (b) Dependence of treewidth on the number of qubits.

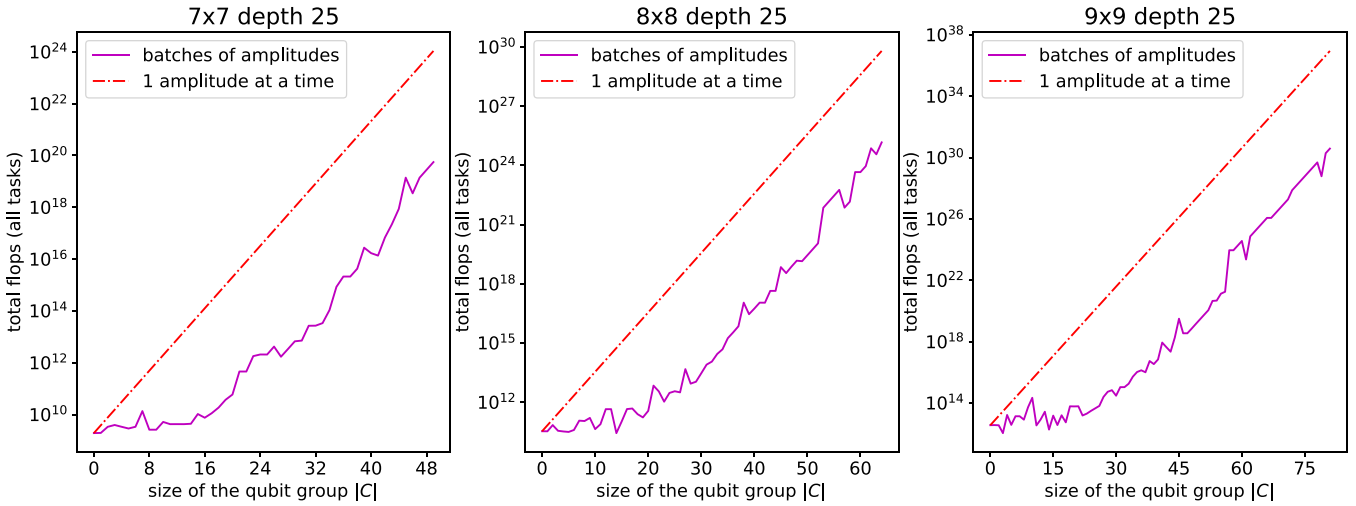

FIG. 9. Total flop requirements for the simulation of a typical random circuit of varying size. Shown is the predicted number of floating-point operations for the simulation of the full subset of amplitudes of $|C|$ qubits (there are $2^{|C|}$ amplitudes). In the case of one amplitude at a time simulation, a combined cost of all tasks is drawn.

the work of Boixo *et al.* The qubits are arranged in a square grid of size $k \times k$, and a set of gates $\{X^{1/2}, Y^{1/2}, cZ, T, H\}$ is applied in a predefined pattern. This circuit choice can be implemented in superconducting quantum processors [32]. The reader is referred to Ref. [10] to learn more details about the motivation of these random circuits. The dataset with random circuits of Boixo *et al.*, which we used in this work, is available online [33]. Here we are interested only in the numerical cost and the memory usage to evaluate amplitudes.

In Fig. 8, the dependence of the treewidth ν on the size and depth of the random circuits is shown. Let us recall that the number of floating-point operations scales as $O(2^{\nu+1})$ and the required memory as $O(2^{\nu})$. The complexity of simulation grows exponentially with the volume of random quantum programs, which was the original motivation for considering them as a test bench for demonstrations of "quantum supremacy" [4].

In Fig. 9, the advantage of batch simulation comparing to single amplitude at a time is shown. The steep growth of the flop cost is significantly ameliorated. We recall that a clique C is introduced into the computational graph when the evaluation of all amplitudes of $|C|$ qubits is performed. While this clique is less than the treewidth of the computational graph, there is only a negligible increase in the computational cost. Batch simulation, however, requires copious amounts of memory, as shown in Fig. 10. The results we obtain illustrate a usual CPU/memory trade-off seen in numerical algorithms. Notice also that the curves for total amount of memory and flop are almost indistinguishable. This result is caused by the fact that during the evaluation of the amplitudes one needs to contract high-order tensors over an index of size 2: the flop cost of the most expensive contraction equals the size of the largest tensor times 2.

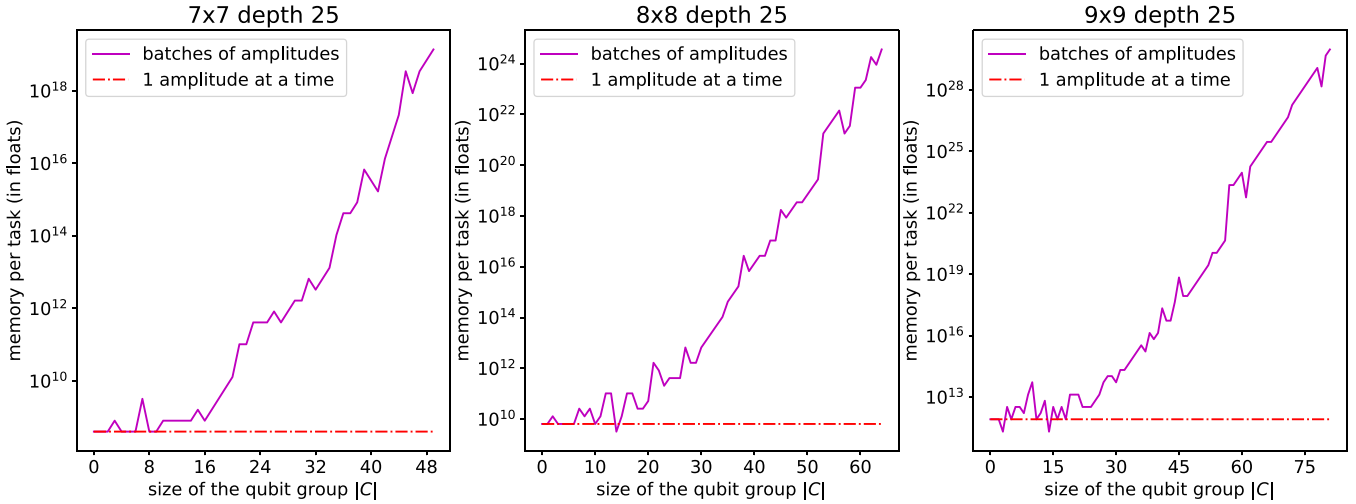

FIG. 10. Minimal memory requirement for the simulation of a typical random circuit of varying size. Shown is the predicted number of memory in floating-point units per single task (simulation of either one amplitude or a batch of amplitudes). Notice the high similarity with Fig. 9: the ratio of flop to memory access is almost constant in logarithmic scale.

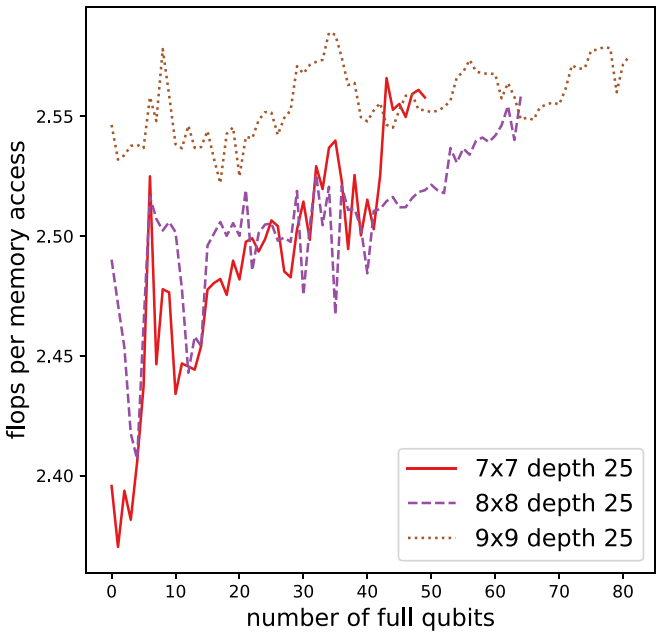

FIG. 11. Floating point to memory access ratio during the simulation of random circuits of varying size.

Last, we provide the dependence of the number of operations per memory access for circuits of different sizes in Fig. 11. In all cases, the values are in the range of $O(L)$, where $L = 2$ for qubits. This dependence demonstrates that despite a potential for massive parallelism [7], the problem of tensor contraction is essentially memory bound, and an efficient algorithm has to very carefully overlap data transmission and computations. Compared to the contraction of matrices, where extremely efficient algorithms were developed [34] using CPU cache and vectorized operations, a general tensor contraction is more challenging for optimization.

V. CONCLUSION AND COMPARISON

We introduced a way to optimize graphical model algorithms for quantum circuit simulation. Our approach allows the user to select between the amount of memory consumed and the speed of the calculation; thus, the code can be adapted to the available hardware. We emphasize that our approach is not restricted to quantum circuit simulation, but can be used to evaluate partial contractions of general tensor networks. This is a method which evaluates partial contractions efficiently; e.g., its resource requirements depend only on the treewidth of the expression's graph.

Many more improvements to the tensor contraction strategy can be proposed. In this article, we explicitly avoid the discussion of parallel simulation and defer it to the upcoming publication. Also, due to the highly heterogenic structure of modern computer memories, some research is needed on the proper scheduling of the operations, especially in the parallel case.

We hope that the discussion in this article highlights a fundamental connection between tensor networks, graphs, and quantum systems.

Algorithm 3. Forming buckets from the expression graph

Input: $G = (V, E)$, G encodes a circuit, $\pi : V \to \mathcal{N}$, $\pi = \{(v_i, i)\}_{i=1}^{|V|}$
Output: $\{B_i\}_{i=1}^{|V|}$

function FORM_BUCKETS(G, π)
  for $i \in [1, \ldots, |V|]$ **do**
    $v \leftarrow \pi^{-1}(i)$
    for T not in $\{B_i\}_{i=1}^{|V|}$ **do**
      if T is indexed by v
        $B_i \leftarrow B_i \cup T$
      end if
    end for
  end for
end function

ACKNOWLEDGMENTS

The authors thank Georgy Ovchinnikov for helpful discussions. This research was supported by the Huawei Company. We kindly thank Xuecang Zhang and Yuri Zotov for their guidance, and we thank Dmitry Kolmakov for the work on the efficient implementation of the algorithm and technical interaction with the team from Huawei's side.

APPENDIX: BUCKET ELIMINATION

As is shown in the main text, graphical models are a convenient way to represent tensor contractions. One of the ways to perform contractions is the bucket elimination algorithm [21,22]. The idea of bucket elimination is simple: one starts with a graph $G = (V, E)$, which corresponds to a tensor network. Given an order $\pi: V \to \{0, \ldots, |V| - 1\}$, we eliminate nodes in V according to π one by one, until all nodes are removed.

The bucket elimination is implemented as follows. First buckets (sets) are formed according to the variable elimination order π. For each variable v (which is also a node in G), we form a set of tensors $B_{\pi(v)}$ which are indexed by v. If both variables v and w index the same tensor T, then T is placed only in the bucket corresponding to a variable with minimal order, e.g., $T \in B_{\min[\pi(v),\pi(w)]}$. The algorithm to form buckets is listed in Algorithm 3.

Having arranged the tensors into the bucket structure, one can proceed to the evaluation of the expression. We process buckets according to their order. For every bucket, all tensors in it are contracted over the bucket's variable. The result is a new tensor, which is an intermediate in the contraction expression. This intermediate tensor is added to the bucket corresponding to its variable v with the lowest order $\pi(v)$. The bucket processing algorithm is listed in Algorithm 4.

It has to be noted that if one would stop the algorithm before all buckets are processed, e.g., when stop_index $< |V|$, then the rest of the buckets $\{B\}_{\text{stop\_index}}^{|V|}$ would contain a *partial* contraction of the original tensor network.

Another interesting use of bucket elimination is for the estimation of the numerical cost of tensor contractions. Indeed, instead of performing the contraction over actual tensors

Algorithm 4. Contracting expression using bucket structure

Input: An ordered set of sets $\{B_i\}_{i=1}^{|V|}$ holding tensors, $\pi = \{(v_i, i)\}_{i=1}^{|V|}$
Output: tensor
1: **function** PROCESS_BUCKETS(B, π, stop_index)
2: result $\leftarrow 1$
3: **for** $i \in [1, \ldots,$ stop_index$]$ **do**
4: $v \leftarrow \pi^{-1}(i)$
5: $T \leftarrow$ contract over v all $\tilde{T} \in B_i$
6: **if** T is scalar **then**
7: result $\leftarrow$ result $\cdot\, T$
8: **else**
9: $k = \pi(w)$, w indexes T, w is minimal w.r.t. π
10: $B_k \leftarrow B_k \cup T$
11: **end if**
12: **end for**
13: **return** result
14: **end function**

inAlgorithm 4, one can count the number of floating-point operations and the amount of used memory with their symbolic representations. Let us consider an example contraction in Eq. (A1), where for simplicity the sizes of all indices are taken to be the same (as is in the case of quantum circuits, where $L = 2$):

$$C_{ijk} = \sum_l A_{ijl} B_{jkl}, \qquad \dim(i) = \dim(j) = \dim(k) = \dim(l) = L. \tag{A1}$$

To evaluate the result of Eq. (A1) one would need L^4 multiplications and additions. The whole expression would require $3L^3$ of storage. The described idea can be used to estimate numerical costs of contracting a tensor network given its graphical model G and an elimination order π.

[1] Intel, 2018 CES: Intel Advances Quantum and Neuromorphic Computing Research (2018).
[2] IBM, IBM Q Experience (2018).
[3] A. W. Harrow and A. Montanaro, Nature (London) **549**, 203 (2017).
[4] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, Nat. Phys. **14**, 595 (2018).
[5] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya *et al.*, Science **360**, 195 (2018).
[6] R. Li, B. Wu, M. Ying, X. Sun, and G. Yang, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 31 (IEEE, 2020), pp. 805–816.
[7] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, arXiv:1805.01450.
[8] P. W. Shor, in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science* (IEEE, New York, 1994), pp. 124–134.
[9] I. L. Markov and Y. Shi, SIAM J. Comput. **38**, 963 (2008).
[10] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, arXiv:1712.05384.
[11] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff, arXiv:1710.05867.
[12] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, T. Lippert, H. Watanabe, and N. Ito, Comput. Phys. Commun. **176**, 121 (2007).
[13] H. De Raedt, F. Jin, D. Willsch, M. Willsch, N. Yoshioka, N. Ito, S. Yuan, and K. Michielsen, Comput. Phys. Commun. **237**, 47 (2019).
[14] S. Aaronson and L. Chen, arXiv:1612.05903.
[15] J. C. Bridgeman and C. T. Chubb, J. Phys. A: Math. Theor. **50**, 223001 (2017).
[16] R. N. C. Pfeifer, J. Haegeman, and F. Verstraete, Phys. Rev. E **90**, 033315 (2014).
[17] Z.-Y. Chen, Q. Zhou, C. Xue, X. Yang, G.-C. Guo, and G.-P. Guo, Sci. Bull. **63**, 964 (2018).
[18] J. Carrasquilla, D. Luo, F. Pérez, A. Milsted, B. K. Clark, M. Volkovs, and L. Aolita, arXiv:1912.11052.
[19] F. Pan, P. Zhou, S. Li, and P. Zhang, arXiv:1912.03014.
[20] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, D. P. Mandic *et al.*, Foundations and Trends® in Machine Learning **9**, 431 (2016).
[21] R. Dechter, in *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence* (Morgan Kaufmann Publishers Inc., San Francisco, CA, 1996), pp. 211–219.
[22] S. Marsland, *Machine Learning: An Algorithmic Perspective* (Chapman & Hall/CRC, London, 2011).
[23] H. L. Bodlaender, Acta Cybernetica **11**, 1 (1994).
[24] L. Chi-Chung, P. Sadayappan, and R. Wenger, Parallel Process. Lett. **7**, 157 (1997).
[25] V. Gogate and R. Dechter, in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence* (AUAI Press, Arlington, Virginia, 2004), pp. 201–208.
[26] H. L. Bodlaender, F. V. Fomin, A. M. Koster, D. Kratsch, and D. M. Thilikos, in *European Symposium on Algorithms* (Springer, New York, 2006), pp. 672–683.
[27] T. Kloks, *Treewidth: Computations and Approximations* (Springer, New York, 1994), Vol. 842.
[28] T. Kloks, H. Bodlaender, H. Müller, and D. Kratsch, in *European Symposium on Algorithms* (Springer, New York, 1993), pp. 260–271.
[29] F. Gavril, SIAM J. Comput. **1**, 180 (1972).
[30] J. R. Blair and B. Peyton, in *Graph Theory and Sparse Matrix Computation* (Springer-Verlag, Berlin, 1993), pp. 1–29.
[31] R. E. Tarjan and M. Yannakakis, SIAM J. Comput. **13**, 566 (1984).
[32] Y. Chen, C. Neill, P. Roushan, N. Leung, M. Fang, R. Barends, J. Kelly, B. Campbell, Z. Chen, B. Chiaro *et al.*, Phys. Rev. Lett. **113**, 220502 (2014).
[33] Random circuits dataset, https://github.com/sboixo/GRCS.git (2019), accessed: 2019-09-16.
[34] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, ACM Trans. Math. Software (TOMS) **14**, 1 (1988).