Constructing Optimal Contraction Trees for Tensor Network Quantum Circuit Simulation

Cameron Ibrahim[†], Danylo Lykov [*], Zichang He [‡], Yuri Alexeev [*], Ilya Safro[†]
[*]Computational Science Division, Argonne National Laboratory, Argonne IL, USA
[†]Department of Computer and Information Sciences, University of Delaware, Newark DE, USA
[‡]University of California Santa Barbara, Santa Barbara, California, USA

***Abstract*—One of the key problems in tensor network based quantum circuit simulation is the construction of a contraction tree which minimizes the cost of the simulation, where the cost can be expressed in the number of operations as a proxy for the simulation running time. This same problem arises in a variety of application areas, such as combinatorial scientific computing, marginalization in probabilistic graphical models, and solving constraint satisfaction problems. In this paper, we reduce the computationally hard portion of this problem to one of graph linear ordering, and demonstrate how existing approaches in this area can be utilized to achieve results up to several orders of magnitude better than existing state of the art methods for the same running time. To do so, we introduce a novel polynomial time algorithm for constructing an optimal contraction tree from a given order. Furthermore, we introduce a fast and high quality linear ordering solver, and demonstrate its applicability as a heuristic for providing orderings for contraction trees. Finally, we compare our solver with competing methods for constructing contraction trees in quantum circuit simulation on a collection of randomly generated Quantum Approximate Optimization Algorithm Max Cut circuits and show that our method achieves superior results on a majority of tested quantum circuits.**

Reproducibility: Our source code and data are available at https://github.com/cameton/HPEC2022_ContractionTrees.

***Index Terms*—Contraction Tree, Tensor Network, Quantum Circuit Simulation, QAOA**

I. Introduction

Tensor networks have become a ubiquitous tool for describing high dimensional, data-sparse tensors in a space efficient manner [8], [15], [30]. In quantum circuit simulation, tensor networks can be used to model circuits with far more qubits than a more direct approach such as state vector simulation, so long as the circuit is sufficiently simple [25]. As such, tensor network based simulation methods will remain vital for the study of quantum algorithms that require a large number of qubits until sufficient advances are made with quantum hardware [25]. Moreover, even with the advanced hardware, such simulation methods will be important for such tasks as finding suitable parameters in variational quantum algorithms that utilize both quantum and classical machines [22]. These networks arise naturally in a variety of disciplines such as Combinatorial Scientific Computing [**?**], Probabilistic Modelling [30], Constraint Satisfaction [20], Machine Learning and Data Mining [12], Physics Modelling [28], and Quantum & Classical Circuit Simulation [15], [27].

A tensor network is most easily conceptualized as a collection of high-order matrices called tensors which are connected by edges. These networks are often used to represent tensors which are impractical to store explicitly in memory [8]. The fundamental operation performed on these networks is called a tensor contraction, where two tensors are combined to create a third tensor. A series of tensor contractions performed on the network is called a tensor network contraction, which is used to reduce the overall size of the network in order to evaluate entries of the underlying tensor that the network represents.

It is useful to represent a tensor network contraction as a contraction tree, a binary tree whose leaves are tensors in the network, where each internal vertex represents the tensor contraction of its two children. In the general case, the cost of any single tensor contraction is exponential in the number of dimensions of its inputs, as is the size of its output [29]. As such, a vital part of performing a tensor network contraction is choosing a contraction tree which minimizes some cost, such as minimizing the size of any intermediary tensor or the total number of operations performed during your tensor network contraction [15]. However, it is NP-hard to construct an optimal contraction tree in the general case for these objectives [29], meaning no optimal solution can been found deterministically in polynomial time.

Our contribution In this paper, we will show how the computationally hard portion of constructing a contraction tree can be reduced to a problem of linear ordering, and give algorithms for constructing an optimal contraction tree for a given ordering. Furthermore, we will give a useful heuristic for choosing orderings which will result in high quality contraction trees. This algorithm is being developed with integration with the QTensor quantum circuit simulation library in mind [23]–[26]. In general, in order to run simulations of quantum circuits with large numbers of qubits, we must be able to find a good quality contraction order. As such, finding optimal or near optimal contraction orders is critical for the development and testing of new quantum algorithms, benchmarking and profiling upcoming quantum devices, and verification of quantum advantage and supremacy claims to name a few applications [3] *Even small improvements to the quality of the contraction tree will lead to a significant acceleration of quantum simulation. We demonstrate an improvement over competitive solvers by orders of magnitude on some instances*

for a comparable running time.

The paper is structured the following way. Section II provides an introduction to tensor networks and linear orderings. Section III introduces existing solvers used for constructing contraction trees for quantum circuit simulation, which we use for benchmarking the proposed algorithm. In Section IV, we present a novel polynomial time algorithm for constructing an optimal contraction tree from a given order and explain the ordering heuristic that was used. In Section V, we evaluate the developed tree structure optimization algorithm against Tamaki-2017, FlowCutter, and Cotengra on a collection of QAOA circuits on different comparison metrics. Finally, Section VI discusses possible areas for future work, while Section VII offers a conclusion.

II. Background and notation

A. Matrix Chains and Tensor Networks

An m by n matrix M is a 2 dimensional array of scalar numbers from some field $\mathbb{F}$ (e.g. $\mathbb{R}$, $\mathbb{C}$). We say this matrix has two indices, i and j, where $\text{size}(i) = m$ and $\text{size}(j) = n$. The size of M is $\text{size}(M) = \text{size}(i)\,\text{size}(j)$.

A matrix M_1 with indices $\{i, j\}$ and a matrix M_2 with indices $\{j, k\}$ can be combined to form a matrix $M_1 * M_2$ with indices $\{i, k\}$ using an operation known as matrix multiplication. Note that the output has the indices of both its inputs with the shared index removed. The number of operations required to perform a matrix multiplication is $\text{size}(i)\,\text{size}(j)\,\text{size}(k)$.

Say we are given matrices $M_1, M_2, \ldots, M_{n-1}, M_n$ with indices $\{i_1, i_2\}, \{i_2, i_3\}, \ldots, \{i_{n-1}, i_n\}, \{i_n, i_{n+1}\}$. The Matrix Chain Ordering Problem aims to find the optimal parenthesization of the matrix chain expression

$$M_1 * M_2 * \cdots * M_{n-1} * M_n$$

which minimizes the cost of computing this product [9]. Note that this setup assumes each matrix shares indices only with its neighbors. A given parenthesization can be expressed as a binary tree with leaves labeled with matrices. For example, the binary tree in Fig. 1 corresponds to the parenthesized matrix chain

$$((A * B) * C) * (D * E).$$

We define tensors analogously. A tensor X is an N-dimensional array of scalar numbers from some field $\mathbb{F}$ with indices $I = \{i_1, \ldots, i_N\}$ (for other perspectives, see [5]).

For a tensor X and an index i, we say that X and i are incident to one another if i is an index of X. We define $\text{inc}(X)$ as the set of indices incident to X and $\text{inc}(i)$ as the set of tensors incident to i. The size of a tensor X is given by

$$\text{size}(X) = \prod_{i \in \text{inc}(X)} \text{size}(i)$$

Two tensors X_1 with indices I_1 and X_2 with indices I_2 can similarly be combined using an operation known as tensor contraction. The output $X_1 * X_2$ will have indices $I_1 \triangle I_2$, where $\triangle$ denotes symmetric difference. That is, the output has indices equal to the union of the input indices, with the intersection removed. $I_1 \triangle I_2$ are known as output indices, while $I_1 \cap I_2$ are known as the shared indices. The number of scalar operations required to perform a tensor contraction is given by

$$\text{ops}(X, Y) = \text{size}(X * Y)\,\text{share}(X, Y)$$

where

$$\text{share}(X, Y) = \prod_{i \in S} \text{size}(i), \qquad S = \text{inc}(X) \cap \text{inc}(Y).$$

A set of tensors $\mathcal{T} = \{X_1, \ldots, X_n\}$ with indices

$$\mathcal{I} = \bigcup_{X \in \mathcal{T}} \text{inc}(X)$$

is known as a tensor network. Defining $\mathcal{T}^{(0)} = \mathcal{T}$, a tensor network contraction iteratively picks $X^{(k)}, Y^{(k)} \in \mathcal{T}^{(k)}$ to produce

$$\mathcal{T}^{(k+1)} = \left(\mathcal{T}^{(k)} \setminus \left\{X^{(k)}, Y^{(k)}\right\}\right) \cup \left\{X^{(k)} * Y^{(k)}\right\},$$

terminating when $|\mathcal{T}^{(k)}| = 1$. A contraction order used for this tensor network contraction can again be represented as a binary tree [29]. For example, the tree shown in Fig. 1 corresponds to the network contraction

$$\begin{aligned}
\mathcal{T}^{(0)} &= \{A, B, C, D, E\} \\
\mathcal{T}^{(1)} &= \{A * B, C, D, E\} \\
\mathcal{T}^{(2)} &= \{(A * B) * C, D, E\} \\
\mathcal{T}^{(3)} &= \{(A * B) * C, D * E\} \\
\mathcal{T}^{(4)} &= \{((A * B) * C) * (D * E)\}.
\end{aligned}$$

Contraction of a tensor network is #P-Hard in the general case [29]. However, many tensor networks arising in practice may be contracted efficiently given a good quality contraction order with respect to some cost function. The costs we consider in this paper are the sum of the number of scalar operations, maximum number of scalar operations, and maximum tensor size, which are defined as follows:

$$\sum_k \text{ops}\left(X^{(k)}, Y^{(k)}\right), \tag{1}$$

$$\max_k \text{ops}\left(X^{(k)}, Y^{(k)}\right), \qquad \max_k \text{size}\left(X^{(k)} * Y^{(k)}\right). \tag{2}$$

The problem of finding an optimal contraction order is itself NP-Hard [29].

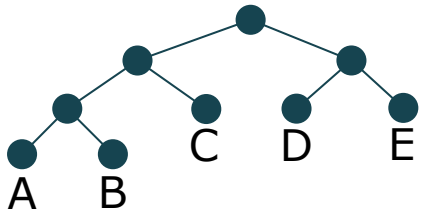

Fig. 1: An example contraction tree with labeled leaves.

While minimizing the number of scalar operations required would seem to be more useful than minimizing the maximum number of operations for any contraction, the latter is actually useful in a parallel setting where many contractions can be performed simultaneously [21].

B. Quantum Circuits and QAOA

An overview to quantum circuits and quantum computing can be found here [43]. A particular type of quantum circuit which has garnered great deal of research interest in recent years is the Quantum Approximate Optimization Algorithm ansatz, particularly those developed to target the graph theoretic MaxCut problem [13]. QAOA is a variational quantum-classical algorithm inspired by the adiabatic evolution principle. It is essentially a quantum annealer with a finite number of steps p. The quality of the final solution increases with p, as does the complexity of the circuit.

In quantum circuit simulation, it is possible to represent a quantum circuit as a high dimensional tensor over the complex numbers [44] or, equivalently, as a tensor network [27]. In this perspective, each quantum gate is associated with a tensor X with indices corresponding to the inputs and outputs of the gate. Two tensors in such a network will share an index only if it corresponds to a value that has been passed from one corresponding gate to the other. For the purposes of this paper, we will assume the size of every index in the network is 2.

With this construction, the costs described in Eq 2 are related to older literature which achieved similar estimations on the computational cost of tensor network contractions via quantities called vertex congestion, edge congestion, and treewidth [27], [29].

In particular, vertex congestion and treewidth are equal to the log of the maximum number of operations needed for any contraction

$$\max_k \log_2 \operatorname{ops}\left(X^{(k)}, Y^{(k)}\right), \tag{3}$$

while edge congestion is equal to the log of the maximum size of any intermediate tensor

$$\max_k \log_2 \operatorname{size}\left(X^{(k)} * Y^{(k)}\right). \tag{4}$$

C. Tensor Networks as Graphs

An undirected graph $G = (V, E)$ with no loops and multi-edges, is a set V of vertices and a set $E \subseteq \binom{V}{2}$ of edges. The weights on the edges of the graph are denoted by the weighting function $w\colon E \to \mathbb{R}_{\geq 0}$. For an edge $uv \in E$, $w(uv)$ denotes its weight. The set of incident edges of a vertex v is the set of edges containing v, denoted $\operatorname{inc}(v) = \{e \in E \mid v \in e\}$. The degree of v is the number of edges incident to v, $\deg(v) = |\operatorname{inc}(v)|$.

Say we have a tensor network $\mathcal{T}$ with indices $\mathcal{I}$ where $|\operatorname{inc}(i)| = 2$ for all $i \in \mathcal{I}$. Then $\mathcal{T}$ admits a representation as a weighted, undirected graph $G = (V, E)$. Let F be the function which assigns tensors to vertices and indices to edges, such that $F(i) = \{F(X), F(Y)\}$ if and only if $i \in \operatorname{inc}(X) \cap \operatorname{inc}(Y)$. The weight function w is defined as $w(F(i)) = \log_2 \operatorname{size}(i)$.

A linear vertex ordering for a graph $G = (V, E)$ is a bijective mapping $\sigma\colon V \to \{1, \ldots, |V|\}$. A linear ordering problem on a graph generally aims to find a linear ordering which minimizes a given cost function, such as the p-Sum objective, which is defined as

$$\left(\sum_{uv \in E} w(uv)|\sigma(u) - \sigma(v)|^p\right)^{1/p}$$

[33]. For the special case where $p = 1$, this problem is known as the Minimum Linear Arrangement Problem [32].

D. Multilevel algorithms and refinement

The multilevel approach is a big class of algorithms actively used in many different areas of scientific computing, optimization, and machine learning [6]. In the context of this work, we briefly describe a version of multilevel algorithms for graph optimization problems [7], [31]. When the graph is large and a fast solution of an optimization problem is required for a specific application, it is often useful to compress the problem by (possibly nonlinearly) aggregating variables into, so called, coarse variables. This is done in a such way that a solution for each coarse variable can be effectively interpolated back to those variables that participated in the aggregation. The entire process of problem coarsening is performed gradually forming a multilevel hierarchy of coarse problems. Each next coarser problem approximates the previous finer problem from which it has been created. Thus, the number of variables at each level of this hierarchy is decreasing which allows to solve them faster that the original problem. When sufficiently small coarsest level is created, the best possible (often exact) solution is computed. The last stage of framework (called uncoarsening) is to gradually solve the problems at each level of coarseness by (1) interpolating the initial solution for the current fine level from the coarser level, and (2) refining the interpolated solution.

If both coarsening and uncoarsening are computed locally (i.e., their complexity is linear in the number of variables) then the entire multilevel framework becomes of linear or nearly-linear complexity assuming that the number of variables at each level is decreasing within a factor of 1.5-2.5. It is important to mention that this approach is different than such one-shot compression approaches as truncated SVD. Such problems on graphs as partitioning [2], [34], various linear orderings [18], [35], and community detection [19], [41] have benefited from the multilevel approaches to mention just few of them [42].

III. Related Work

A. FlowCutter

FlowCutter is used collectively to refer to an algorithm for computing small, balanced s-t cuts using Pareto optimization [17], as well as an algorithm utilizing the former to construct a tree decomposition of the input graph via recursive bisection [39], that is also a form of a multilevel algorithm.

FlowCutter was entered into heuristic track of the PACE 2017 Parameterized Algorithms and Computational Experiments Challenge on minimizing treewidth, where it placed second as the only solver to find a solution for all test instances in the allotted 30 minutes [10], [39]. Moreover, FlowCutter is known to be useful in finding high quality tensor network contraction [11].

B. Tamaki-2017

The Tamaki-2017 solver is a tree decomposition solver with components submitted to the exact and heuristic tracks for PACE 2017 [10]. On the heuristic track, this solver ranked first place, tending to achieve better results on smaller instances than the competitive approaches, while sometimes failing to find an answer in the allotted time for larger instances [39].

The Tamaki-2017 algorithm is based on positive-instance driven dynamic programming, and, similar to FlowCutter, operates recursively on minimal separators of the instance graph [40].

C. Cotengra

Cotengra is a library for the efficient contraction of tensor networks [16]. In their seminal paper on the subject, the authors introduce a method for directly constructing contraction trees based on recursive bisection [15]. By utilizing existing hypergraph partitioning solvers such as KaHyPar [36], Cotengra is able to construct contraction trees utilizing the hypergraph structure of some inputs that arise in Quantum Circuit Simulation [15]. Cotengra also utilizes the Bayesian optimization strategy, varying the hyperparameters of the partitioning solver at various levels in order to account for a changing graph structure as the algorithm progresses [15].

D. cuTENSOR

cuTENSOR is a tensor network CUDA library in development by NVidia, which aims to bring high performance tensor primatives to the GPU [1]. Because not much information about their specific method for determining contraction orders is publicly available at the time of writing, we will not be comparing to cuTENSOR at this time.

IV. Constructing a Contraction Tree

Our approach is founded on the observation that a parenthesization of a matrix chain expression or a contraction order for a tensor network both admit representations as binary trees with labelled leaves.

It is useful to introduce a more direct tensor variant of the Matrix Chain Ordering Problem, known as the Tensor Chain Ordering Problem. Given tensors $X_1, \ldots, X_n$, find an optimal parenthesization of the tensor chain expression

$$X_1 * \cdots * X_n$$

which minimizes one of the costs introduced in Eqs 1,2. Notably, indices are not restricted to immediate neighbors in the chain.

Given a binary tree T representing a contraction order O for a tensor network $\mathcal{T}$, consider the linear ordering σ of tensors corresponding to a left to right traversal of the labelled leaves of T. Then T could be considered a possible solution to the Tensor Chain Ordering Problem for the chain

$$\sigma^{-1}(1) * \cdots \sigma^{-1}(n).$$

Then finding a contraction order for a tensor network can actually be reframed as finding a linear ordering of the tensors in the network, then solving the Tensor Chain Order Problem.

$$\min_{O} \text{cost}(\mathcal{T}, O) = \min_{\sigma} \min_{T} \text{cost}(\mathcal{T}, \sigma, T).$$

We call solving the Tensor Chain Order Problem for a fixed linear order σ tree structure optimization. Notably, an optimal contraction order of the Tensor Chain Order Problem can be found deterministically in polynomial time as demonstrated in the following section. This decouples the actual computationally Hard task of finding an optimal order of the chain, from the more tractable task of actually constructing the tree.

A. Tree Structure Optimization

In this section, we introduce a simple dynamic programming solution to the Tensor Chain Order Problem. Say we are given the tensor chain expression $X_1 * \cdots * X_n$. Let $X_{i,j} = X_i * X_{i+1} * \cdots * X_j$ for $i < j$.

From this, we find the following recursion for computing the minimal edge congestion of any parenthesization of the tensor chain

$$\begin{aligned} c(i,i) &= \text{size}(X_i) \\ c(i,j) &= \min_{i \le k < j} \max \begin{cases} \log_2 \text{size}(X_{i,k} * X_{k+1,j}) \\ c(i,k) \\ c(k+1,j) \end{cases} \end{aligned} \tag{5}$$

as well as the minimal vertex congestion

$$\begin{aligned} c(i,i) &= \text{size}(X_i) \\ c(i,j) &= \min_{i \le k < j} \max \begin{cases} \log \text{ops}(X_{i,k}, X_{k+1,j}) \\ c(i,k) \\ c(k+1,j) \end{cases} \end{aligned} \tag{6}$$

It is beneficial to avoid attempting to calculate $\text{share}(X_{i,k}, X_{k+1,j})$ at each iteration. Instead, we introduce the subroutine Alg. 1 which more efficiently calculates every value of shared as k is varied across the sequence. If $d = \max_{X \in \mathcal{T}} |\text{inc}(X)|$ is the maximum degree of any tensor in $\mathcal{T}$, this can be done in $O(d|\mathcal{T}|)$ time.

From these recursions, we introduce Alg. 2, a dynamic programming algorithm for computing the minimal possible size of the largest tensor for any contraction tree given an order. We make use of pruning in order to reduce the number of needed computations, as there is no need to evaluate the second subsequence in a split if the first has already exceeded the current best result. Performance can be further improved by making use of memoization to cache previous calls. Additional gains may be achieved by parallelizing the evaluation of disjoint subproblem calls, which is an ongoing project. Utilizing these techniques, the complexity of Alg. 2 is $O(|E||V|^2 + d|V|^3)$.

Algorithm 1 CalcShared

Require: A tensor chain $C = (X_1, \ldots, X_n)$, $1 \leq i \leq j \leq n$
  $shared \leftarrow [0 \mid \text{for } i \leq k < j]$
  $(upper, lower) \leftarrow (\{X_i, \ldots, X_{k-1}\}, \{X_{k+1}, \ldots, X_j\})$
  for $i \leq k < j$ **do**
    $new \leftarrow sum(\text{size}, \{a \in \text{inc}(X_i) \mid \text{inc}(a) \cap upper \neq \emptyset\})$
    $old \leftarrow sum(\text{size}, \{b \in \text{inc}(X_i) \mid \text{inc}(b) \cap lower \neq \emptyset\})$
    $shared[k] \leftarrow shared[k] + new - old$
    $shared[k+1] \leftarrow shared[k]$
  end for
  return $shared$

Algorithm 2 Tree Structure Optimization

Require: A tensor chain $C = (X_1, \ldots, X_n)$, $1 \leq i \leq j \leq n$
Require: $x \oplus y = x + y$ or $\max(x, y)$
  if $i == j$ **then**
    return $(\text{size}(X_i), i)$
  end if
  $shared \leftarrow CalcShared(C, i, j)$
  $outsize \leftarrow \text{size}(X_{i,j})$
  $(c^*, t^*) \leftarrow \infty, i$
  for $i \leq k < j$ **do**
    $c \leftarrow COST(outsize, shared[k])$
    if $c > c^*$ **then**
      continue
    end if
    $(c_l, l) \leftarrow TreeStructureOptimization(C, i, k)$
    $c \leftarrow c \oplus c_l$
    if $c > c^*$ **then**
      continue
    end if
    $(c_r, r) \leftarrow TreeStructureOptimization(C, k+1, j)$
    $c \leftarrow c \oplus c_r$
    if $c < c^*$ **then**
      $(c^*, t^*) \leftarrow (c, (l, r))$
    end if
  end for
  return (c^*, t^*)

A variant for the classical total number of operations can be achieved simply by looking at the incremental sum of the left and right trees rather than the max.

$$\begin{aligned} c(i,i) &= \text{size}(X_i) \\ c(i,j) &= \min_{i \leq k < j} \text{ops}(X_{i,k}, X_{k+1,j}) + c(i,k) + c(k+1,j) \end{aligned} \tag{7}$$

B. Heuristic Order Choices

While we have introduced algorithms for constructing an optimal contraction tree given a particular order, we must now tackle the problem of actually choosing an order. We turn to a heuristic approach for solving this problem. In particular, we select an order which minimizes the sum total of lengths of stretched edges in the graph. This is known as the Minimal Linear Arrangement of the vertices in the graph, and is defined as

$$\sum_{uv \in E} w(uv)|\sigma(u) - \sigma(v)|$$

This is motivated by the property of matrix chains sharing indices only with their immediate neighbors. An order which minimizes MLA discourages long stretched edges on the chain. We have empirically observed this to be effective in reducing the congestion of the resulting contraction tree, which will be discussed further in the Results section.

In order to calculate an order which effectively minimizes MLA, we utilize the LinearOrdering.jl package, a Julia package for multilevel linear ordering that is currently in development by the authors. This package utilizes a volume based coarsening strategy as well as node by node minimization in order to find a high quality arrangement [32]. The package can be found at https://github.com/cameton/LinearOrdering.jl.

V. Experimental Results

We evaluate tree structure optimization against existing state of the art algorithms Tamaki-2017, FlowCutter, and Cotengra on a collection of randomly generated QAOA circuits. To generate test results for tree structure optimization, we find an order for the graph with a good Minimum Linear Arrangement objective, then run tree structure optimization with the relevant cost. This is done repeatedly while varying the random seed for a set amount of time, returning the best result at the end. In most cases we observe a significant improvement in the quality of results given a comparable running time (that is usually negligible in comparison to the quantum simulation time itself).

For our tests, for a fixed depth p and degree d, we construct a p layer QAOA ansatz circuit targeting the Max Cut problem on a random 32 vertex d-regular graph using the QTensor library [23]. Each circuit will have a number of qubits equal to the number of vertices in the given graph. For each combination of $d = 3, 4, 5$ and QAOA depth $p = 2, 3, 4, 5$, we generate 10 random circuits to form our test set. The source code and generated circuits are available at https://github.com/cameton/HPEC2022_ContractionTrees. In doing so, we sample a variety of circuits with a varying level of connectedness and depth, which we use as a proxy for the complexity of the circuit; such circuits are commonly used in evaluations of contraction order solvers for quantum circuits as in [15].

A. Vertex Congestion

We compare the vertex congestion (Eq. 3) of the solution found by our tree structure optimizer for the test set against that found by Tamaki-2017 and FlowCutter by finding the tree decomposition of the line graph. As Cotengra has functionality to minimize edge congestion and number of FLOPS, it will be evaluated separately. For these experiments, we ran each of these optimizers for 5 seconds. Results for this test are given in Fig. 2.

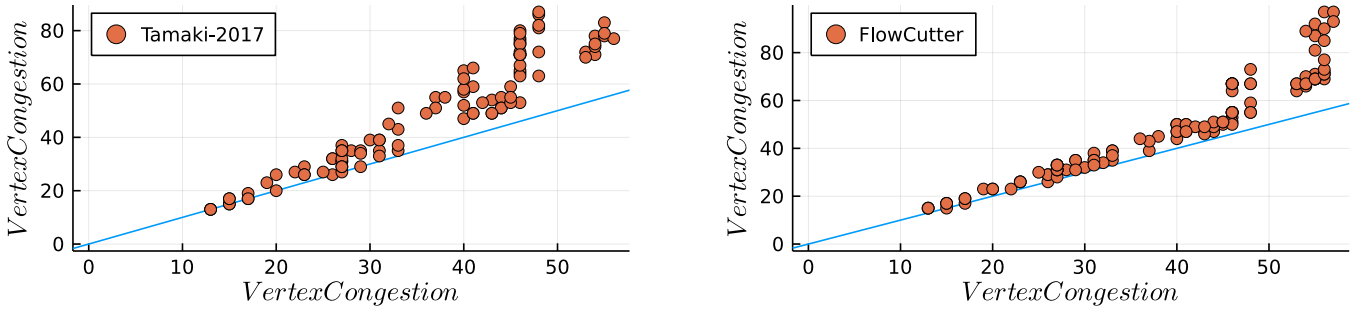

Fig. 2: Vertex Congestion achieved by Tamaki-2017 (left) and FlowCutter (right) plotted against the vertex congestion found by our tree structure optimization on a Minimum Linear Arrangement order. The blue line represents the threshold at which both solvers achieved the same objective value. Points above this threshold indicate that tree structure optimization found a superior value.

Our tree structure optimization based approach performed strictly better than Flowcutter on 96.4% of points and better than Tamaki on 90.0% of points, while never achieving a worse cost. What's more, the margin of improvement increases significantly as the vertex congestion of the circuit increases.

As vertex and edge congestion represent the log of the maximum size and maximum number of operations, even small improvements in congestion lead to significant gains. As such, having such a large improvement on so many points represents advantage of our solver over competitive solvers run for the same amount of time.

B. Edge Congestion and Operation Count

We compare the edge congestion (Eq. 4) and estimated number of operations (Eq. 1) of the solution found by our tree structure optimizer on the test set against those found by Cotengra. For these tests, Cotengra was run for 5 seconds with default settings and the KaHyPar partitioner backend. Two tests were run using Cotengra: one minimizing edge congestion and one minimizing the total number of scalar operations. Results are given in Fig. 3.

From Fig. 3, we see that tree structure optimization is highly competitive with Cotengra, in every case finding a solution which is close to or better than Cotengra in terms of both estimated number of FLOPS and edge congestion.

Our solver bests Cotengra on a full 95% of points when minimizing total number of operations, and on 84.1% of points when minimizing edge congestion. This once again demonstrates the superiority of our solver for comparatively short run times.

C. Varying the Number of Qubits

We may also wish to compare these solvers as we vary the number of vertices in the input graph, by extension varying the number of qubits. To avoid redundancy, we restrict our comparison for this section to Cotengra and Total FLOPS alone. For these tests, we construct a 2 layer QAOA ansatz circuit targeting the Max Cut problem for random 3-regular graphs with $32, 64, 96$ and 128 vertices. For each size, we generate 10 random circuits. Because each circuit has a number of qubits equal to the size of the input graph, we can examine circuits of a similar complexity across a variety of different qubit counts. For these tests, we ran both Cotengra and the ordering solver for 10 seconds for each input.

The results of this test are shown in Fig. 4. Averaging over the 32 qubit circuits, our tree structure optimization based approach produced results around 28.6% better than Cotengra, improving to 74.1% when averaging over the 64 qubit circuits. For the 96 and 128 qubit circuits, nearly every result produced by our tree structure optimization approach is several orders of magnitude better than Cotengra for the 10 second running time.

VI. Future Work

A. Width Refinement

The current refinement strategy used in this paper minimizes Minimum Linear Arrangement by making local changes to the order at each level of the algorithm [33]. While we have shown thus far that Minimum Linear Arrangement works as a effective heuristic for the tested circuits, we may be able to achieve further improvements through the development of refinement strategies which directly target the width of the order. That is, we would like the ability to make local changes which serve to reduce the width of the overall order, potentially achieving better results than the more general Minimum Linear Arrangement heuristic.

B. Performance & Scalability

The algorithms introduced here for tree structure optimization are currently not parallelized. The development of a tree structure optimization algorithm which takes advantage of the graph structure for improved performance is vital to the application of this technique to fields outside of quantum circuit simulation, such as in large scale shortest path acceleration [17].

Although the multilevel algorithm component is already fast, it will be valuable to develop a strongly parallelized version of these algorithms which may be better suited for a high performance computing environment, particularly where such hardware is being used for circuit simulation already.

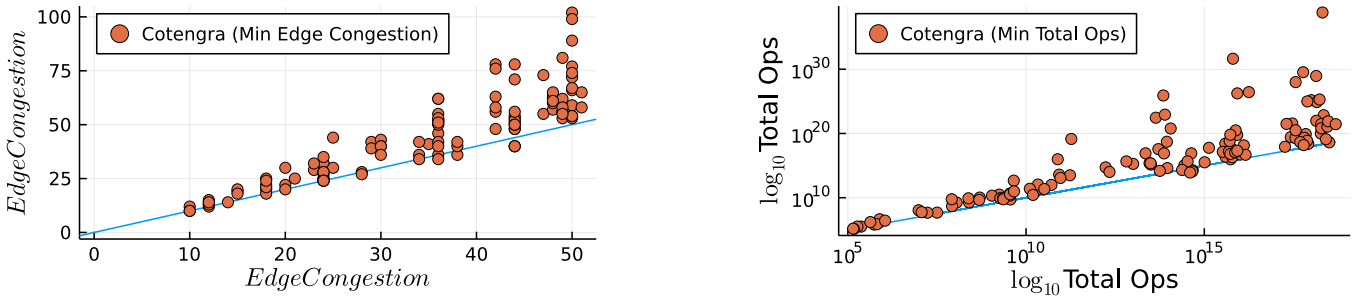

Fig. 3: Edge congestion of Cotengra minimizing edge congestion against tree structure optimization (left) and Cotengra minimizing operation count against tree structure optimization on a log-log scale (right). The blue line represents the threshold at which both solvers achieved the same objective value. Points above this threshold indicate that tree structure optimization found a superior value.

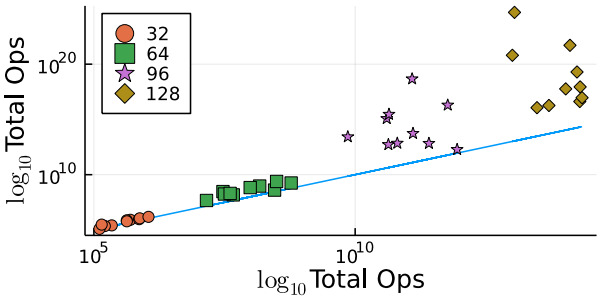

Fig. 4: Cotengra minimizing operation count against tree structure optimization on a log-log scale for circuits with 32, 64, 96, and 128 qubits. The blue line represents the threshold at which both solvers achieved the same objective value. Points above this threshold indicate that tree structure optimization found a superior value.

C. Higher Order Structures

Thus far we have primarily considered tensor networks representable as normal undirected graphs. In this context, we associate each index in the circuit with a corresponding edge in the graph. For some quantum circuits, a possible cost-saving technique is to reconsider certain collections of indices as hyperindices, in essence a single index connecting multiple gates [24]. This introduces a hypergraph structure to our tensor network representation, a higher order generalization of undirected graphs.

As it stands, our work has yet to be extended to accommodate these higher order structures, something which Cotengra is currently able to handle by making use of existing hypergraph partitioning software [15]. In particular, attention needs to be paid to how the hypergraph structure may inform the coarsening portion of the ordering problem. For example, this can be done by evaluating various kernels and similarity measures on high order structures such as in [4], [14], [37], [38]. Moreover, this necessitates a tree structure optimization algorithm for higher order structures.

VII. Conclusion

Accelerating quantum circuit simulation is a critical problem not only for demonstrating quantum advantage but also for hybrid quantum-classical algorithms which require computationally heavy parameter optimization backend on the classical machine. In this paper, we presented novel algorithms for constructing an optimal contraction tree from a given order. We introduced a multilevel solver for the Minimum Linear Arrangement problem on graphs and demonstrated its applicability as a heuristic for providing orderings for contraction trees. We compared the performance of our solver against state-of-the-art industry solvers Tamaki-2017, FlowCutter, and Cotengra on a collection of randomly generated QAOA circuits. We have shown that our method achieves results that are superior by orders of magnitude on some instances to competitors when run for a comparable amount of time. There is work under way to produce higher quality orders for producing contraction trees, and on improving the speed of tree structure optimization.

Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) ONISQ program under Contract No. HR001120C0068. This research was partially supported by NSF, under award number 2122793. Y.A.'s and D.L.'s work at Argonne National Laboratory was partially supported by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

References

[1] cuTENSOR. https://docs.nvidia.com/cuda/cutensor/index.html, 2022.

[2] Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10–pp. IEEE, 2006.

[3] Yuri Alexeev, Dave Bacon, Kenneth R Brown, Robert Calderbank, Lincoln D Carr, Frederic T Chong, Brian DeMarco, Dirk Englund, Edward Farhi, Bill Fefferman, et al. Quantum computer systems for scientific discovery. *PRX Quantum*, 2(1):017001, 2021.

[4] Lu Bai, Edwin R Hancock, and Peng Ren. A jensen-shannon kernel for hypergraphs. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 181–189. Springer, 2012.

[5] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006*, 2017.

[6] A. Brandt and D. Ron. Chapter 1 : Multigrid solvers and multilevel optimization strategies. In J. Cong and J. R. Shinnerl, editors, *Multilevel Optimization and VLSICAD*. Kluwer, 2003.

[7] Cédric Chevalier and Ilya Safro. Comparison of coarsening schemes for multilevel graph partitioning. *Learning and Intelligent Optimization*, pages 191–205, 2009.

[8] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2009.

[10] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration. In Daniel Lokshtanov and Naomi Nishimura, editors, *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*, volume 89 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:12, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[11] Jeffrey M Dudek, Leonardo Duenas-Osorio, and Moshe Y Vardi. Efficient contraction of large tensor networks for weighted model counting through graph decompositions. *arXiv preprint arXiv:1908.04381*, 2019.

[12] Stavros Efthymiou, Jack Hidary, and Stefan Leichenauer. Tensornetwork for machine learning. *arXiv preprint arXiv:1906.06329*, 2019.

[13] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[14] Agata Fronczak, Janusz A Hołyst, Maciej Jedynak, and Julian Sienkiewicz. Higher order clustering coefficients in barabási–albert networks. *Physica A: Statistical Mechanics and its Applications*, 316(1-4):688–694, 2002.

[15] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, mar 2021.

[16] Johnny Gray. Cotengra, 2020.

[17] Michael Hamann and Ben Strasser. Graph bisection with pareto optimization. *ACM J. Exp. Algorithmics*, 23, feb 2018.

[18] Y. F. Hu and J. A. Scott. A multilevel algorithm for wavefront reduction. *SIAM J. Sci. Comput.*, 23(4):1352–1375, 2001.

[19] Isa Inuwa-Dutse, Mark Liptrott, and Ioannis Korkontzelos. A multilevel clustering technique for community detection. *Neurocomputing*, 441:64–78, 2021.

[20] Stefanos Kourtis, Claudio Chamon, Eduardo Mucciolo, and Andrei Ruckenstein. Fast counting with tensor networks. *SciPost Physics*, 7(5), nov 2019.

[21] H. Lee, J. Kim, S.J. Hong, and S. Lee. Processor allocation and task scheduling of matrix chain products on parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):394–407, 2003.

[22] Xiaoyuan Liu, Anthony Angone, Ruslan Shaydulin, Ilya Safro, Yuri Alexeev, and Lukasz Cincio. Layer VQE: A variational approach for combinatorial optimization on noisy quantum computers. *IEEE Transactions on Quantum Engineering*, 3:1–20, 2022.

[23] Danylo Lykov. QTensor. https://github.com/danlkv/qtensor, 2021.

[24] Danylo Lykov and Yuri Alexeev. Importance of diagonal gates in tensor network simulations. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 447–452. IEEE, 2021.

[25] Danylo Lykov, Angela Chen, Huaxuan Chen, Kristopher Keipert, Zheng Zhang, Tom Gibbs, and Yuri Alexeev. Performance evaluation and acceleration of the qtensor quantum circuit simulator on gpus. In *2021 IEEE/ACM Second International Workshop on Quantum Computing Software (QCS)*, pages 27–34. IEEE, 2021.

[26] Danylo Lykov, Roman Schutski, Alexey Galda, Valerii Vinokur, and Yurii Alexeev. Tensor network quantum simulator with step-dependent parallelization. *arXiv preprint arXiv:2012.02430*, 2020.

[27] Igor L Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.

[28] Simone Montangero, Montangero, and Evenson. *Introduction to Tensor Network Methods*. Springer, 2018.

[29] Bryan O'Gorman. Parameterization of tensor network contraction. *arXiv preprint arXiv:1906.00013*, 2019.

[30] Elina Robeva and Anna Seigal. Duality of graphical models and tensor networks. *Information and Inference: A Journal of the IMA*, 8(2):273–288, 2019.

[31] Dorit Ron, Ilya Safro, and Achi Brandt. Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423, 2011.

[32] Ilya Safro, Dorit Ron, and Achi Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[33] Ilya Safro, Dorit Ron, and Achi Brandt. Multilevel algorithms for linear ordering problems. *ACM J. Exp. Algorithmics*, 13, feb 2009.

[34] Ilya Safro, Peter Sanders, and Christian Schulz. Advanced coarsening schemes for graph partitioning. *ACM Journal of Experimental Algorithmics (JEA)*, 19:2–2, 2015.

[35] Ilya Safro and Boris Temkin. Multiscale approach for the network compression-friendly ordering. *J. Discrete Algorithms*, 9(2):190–202, 2011.

[36] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. High-quality hypergraph partitioning. *arXiv preprint arXiv:2106.08696*, 2021.

[37] Ruslan Shaydulin, Jie Chen, and Ilya Safro. Relaxation-based coarsening for multilevel hypergraph partitioning. *SIAM Multiscale Modeling & Simulation*, 17(1):482–506, 2019.

[38] Ruslan Shaydulin and Ilya Safro. Aggregative Coarsening for Multilevel Hypergraph Partitioning. In Gianlorenzo D'Angelo, editor, *17th International Symposium on Experimental Algorithms (SEA 2018)*, volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 2:1–2:15, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[39] Ben Strasser. Computing tree decompositions with flowcutter: Pace 2017 submission. *arXiv preprint arXiv:1709.08949*, 2017.

[40] Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019.

[41] Hayato Ushijima-Mwesigwa, Ruslan Shaydulin, Christian F. A. Negre, Susan M. Mniszewski, Yuri Alexeev, and Ilya Safro. Multilevel combinatorial optimization across quantum architectures. *ACM Transactions on Quantum Computing*, 2(1), February 2021.

[42] C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals Oper. Res.*, 131:325–372, 2004.

[43] John Watrous. Guest column: An introduction to quantum information and quantum circuits 1. *SIGACT News*, 42(2):52–67, jun 2011.

[44] Noson S Yanofsky. An introduction to quantum computing. In *Proof, computation and agency*, pages 145–180. Springer, 2011.