Fast Search of the Optimal Contraction Sequence in Tensor Networks

Ling Liang, Jianyu Xu, Lei Deng, *Member*, *IEEE*, Mingyu Yan, Xing Hu, Zheng Zhang, *Member*, *IEEE*, Guoqi Li, *Member*, *IEEE*, Yuan Xie, *Fellow*, *IEEE*

Abstract—**Tensor network and tensor computation are widely applied in scientific and engineering domains like quantum physics, electronic design automation, and machine learning. As one of the most fundamental operations for tensor networks, a tensor contraction eliminates the sharing orders among tensors and produces a compact sub-network. Different contraction sequence usually yields distinct storage and compute costs, and searching the optimal sequence is known as a hard problem. Prior work have designed heuristic and fast algorithms to solve this problem, however, several issues still remain unsolved. For example, the data format and data structure are not efficient, the constraints during modeling are impractical, the search of the optimal solution might fail, and the search cost is very high. In this paper, we first introduce a log_k order representation and design an adjacency matrix-based data structure to efficiently accelerate the search of the optimal contraction sequence. Then, we propose an outer product pruning method with acceptable overhead to reduce the search space. Finally, we use a multithread optimization in our implementation to further improve the execution performance. We also present in-depth analysis of factors that influence the search time. This work provides a full-stack solution for optimal contraction sequence search from both high-level data structure and search algorithm to low-level execution parallelism, and it will benefit a broad range of tensor-related applications.**

Keywords: Tensor Contraction, Adjacency Matrix, BFS algorithm, Search Space Reduction, Multithread Optimization

I. Introduction

Tensor networks are widely applied in a wide range of applications. The most well-known fields are many-body quantum physics [1]–[5], matrix product states and projected entangled pair states [6]–[11], multiscale entanglement renormalization ansatz [12], [13], and quantum circuit design [14], [15]. Besides the applications in quantum physics, tensor networks are recently applied in IC modeling [16] and EDA problems [17]. In addition, tensor networks are capable of compressing the large-size parameters or data in neural networks [18]–[20] or signal processing algorithms [21], [22].

Tensor contraction, a process of computing a tensor network by eliminating the sharing orders among pairs of tensors, is one of the most fundamental operations in tensor network processing [23]. In a tensor network, the contraction operation iteratively merges two nodes into one until the whole network cannot be merged anymore. Different contraction sequence can result in distinct memory and compute costs. Therefore, finding the optimal contraction sequence which consumes less compute or storage resources is critical for reducing the consequent contraction cost.

However, this might be a very hard problem. On one hand, to find the contraction sequence with optimal compute cost is proved to be NP-hard in [24]. On the other hand, for any tensor network, the optimal storage cost of contraction sequences equals the treewidth of its line graph, which has been proved by [25]. Since the problem of computing the treewidth of a graph is NP-hard in general, it is a rational hyphothesis that the problem of finding the contraction sequence with optimal storage cost is also a hard problem. Therefore, designing heuristic search algorithms seems the only way to find the contraction sequence with optimal storage or compute cost. There exist both depth-first constructive search (DFS) algorithms [24] and breadth-first constructive search (BFS) algorithms [26], [27] to do this search. Dynamic programming can also be applied to solve this problem [27], [28]. These techniques go through all possible contraction sequences before determining the optimal one.

However, the search space grows exponentially as the network size increases. Some algorithms have also been investigated in order to shrink the original search space. For the storage cost, an optimization algorithm is proposed in [29], but this method does not guarantee an optimal sequence. For the compute cost, cost capping is used to prune the sequences that cost more than the optimal one, and some outer product constraints are added to further reduce the search space [27]. However, the search complexity depends on the variance of the sharing orders between tensors. In addition, some algorithms have been proposed to accelerate the search of contraction sequences in closed tensor networks (i.e. the tensor network being contracted into a scalar) [30]. A polynomial search solution is proposed by [31] for considering both storage and compute costs, however, their solution is only effective for the tree structure.

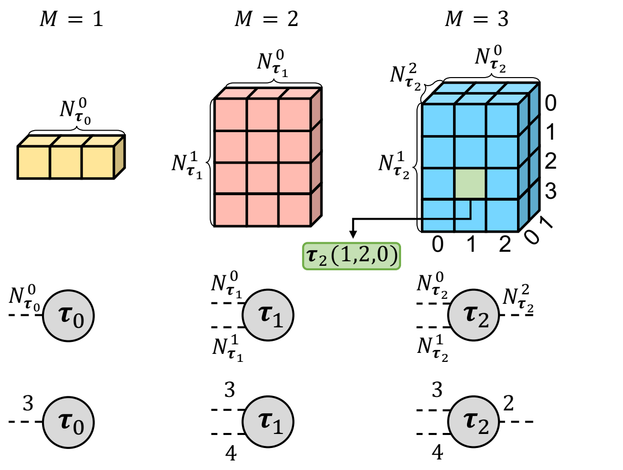
Based on the observations from prior work, several issues

This work was partially supported by National Science Foundation (Grant No. 1817037). Corresponding author: Lei Deng. Ling Liang, Jianyu Xu, Xing Hu, Zheng Zhang, and Yuan Xie are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA (email: {lingliang, xu_jy15, xinghu, yuanxie}@ucsb.edu, zhengzhang@ece.ucsb.edu). Guoqi Li is with the Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing 100084, China (email: liguoqi@mail.tsinghua.edu.cn). Mingyu Yan is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China (email: yanmingyu@ict.ac.cn), and also with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA. Lei Deng is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA (email: leideng@ucsb.edu), and also with the Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing 100084, China.

Table I: Variable definition.

Variable	Definition	Variable	Definition
$\boldsymbol{\tau}_i$	An original single tensor from a tensor network	$\mathcal{T}_I$	A tensor after contraction, where I is a set of i to denote the subscripts of original single tensors
$FO_{\mathcal{T}_I}$	Free order collection of tensor $\mathcal{T}_I$	$SO_{\mathcal{T}_I\mathcal{T}_J}$	Sharing order collection between tensors $\mathcal{T}_I$ and $\mathcal{T}_J$, where $I \cap J = \phi$
$SE_{\mathcal{T}_I}$	Storage expense of tensor $\mathcal{T}_I$	$CE_{\mathcal{T}_I\mathcal{T}_J}$	Compute expense for contracting tensors $\mathcal{T}_I$ and $\mathcal{T}_J$, where $I \cap J = \phi$
MS	Maximum storage expense of a contraction sequence	MC	Maximum compute expense of a contraction sequence
$R_{\mathcal{T}_I}$	Row vector of possible tensor $\mathcal{T}_I$	$O_{\mathcal{T}_I}$	Outer product vector used in the outer product pruning for possible tensor $\mathcal{T}_I$
$\mathcal{T}_{I_1}, \mathcal{T}_{I_2}$	Two split source tensors to contract possible tensor $\mathcal{T}_I$	sq	A contraction sequence
C_V^v	Binomial coefficient	Set_v	A set of possible tensors that are contracted by v original tensors

should be considered in the search of the optimal contraction sequence. First, since the search space is vast, efficient data format and data structure indeed matter and should be designed to accelerate the search process. Second, we should propose an algorithm which can find the optimal solution based on the data structure, shrink the search space, and fit general tensor networks without specific structure constraints. At last, the search time should be superior. In order to make the optimal contraction search more efficient, we propose the following techniques. (1) We design a search algorithm based on the adjacency matrix structure which is friendly to data access and network update. (2) Since the outer product between two tensors can be pruned from the search space, we design an efficient algorithm to identify the prunable tensors. (3) Finally, we adopt multithread optimization for parallel execution of our search algorithm, which can further improve the efficiency. [1] Our proposed method will benefit a broad range of applications that rely on tensor computation.



$M = 1$
$M = 2$
$M = 3$
$N^0_{\boldsymbol{\tau}_0}$
$N^0_{\boldsymbol{\tau}_1}$
$N^1_{\boldsymbol{\tau}_1}$
$N^0_{\boldsymbol{\tau}_2}$
$N^2_{\boldsymbol{\tau}_2}$
$N^1_{\boldsymbol{\tau}_2}$
0 1 2 3
0 1 2
0 1
$\boldsymbol{\tau}_2(1,2,0)$
$\boldsymbol{\tau}_0$
$\boldsymbol{\tau}_1$
$\boldsymbol{\tau}_2$
3
4
2

Figure 1: Examples of tensors with different order configuration. The top subfigures visualize the original tensors; the middle and bottom subfigures show the graph representation of tensors.

II. Preliminaries

In this section, we first introduce the background of tensor, tensor contraction, and the problem definition of finding the optimal contraction sequence of a tensor network in Section II-A. Then, in Section II-B, we describe the vanilla BFS search algorithm that we adopt as the basis of our algorithm design.

A. Tensor Network Contraction

Tensor. The variables commonly used in this paper are listed in Table I. We define a tensor in a network as $\boldsymbol{\tau}_i$. Tensor can be regarded as generalization of vector and matrix to represent high-order data. The number of orders in a tensor is denoted as M, and the length of the m-th order is denoted as $N^m_{\boldsymbol{\tau}_i}$, where $M, N^m_{\boldsymbol{\tau}_i} \in Z^+$. Any element in a tensor can be represented as $\boldsymbol{\tau}_i(n^0_{\boldsymbol{\tau}_i}, n^1_{\boldsymbol{\tau}_i}, ..., n^{M-1}_{\boldsymbol{\tau}_i})$ where we have $n^m_{\boldsymbol{\tau}_i} \in \{0, 1, ..., N^m_{\boldsymbol{\tau}_i} - 1\}$.

In Figure 1, we show the tensor format of vector, matrix and cube. We take the cube as an example, which is is a third-order tensor, i.e. $M = 3$. In this example the length of orders are $N^0_{\boldsymbol{\tau}_2} = 3$, $N^1_{\boldsymbol{\tau}_2} = 4$, and $N^2_{\boldsymbol{\tau}_2} = 2$. We highlight an element in $\boldsymbol{\tau}_2$ which can be denoted as $\boldsymbol{\tau}_2(1, 2, 0)$. We can also use graph representation to denote a tensor, where a vertex is a tensor and an edge stands for one of its orders.

Tensor Contraction under log_k Order Representation. Figure 2(a) shows an example of the tensor contraction between two tensors. In this example, $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$ are two third-order tensors. We use $\mathcal{T}_I$ to represent a tensor after contraction, where I denotes the subscript set of involved original single tensors, e.g. here $\mathcal{T}_I = \mathcal{T}_{01}$. In this example, $\mathcal{T}_{01}$ is a two-order tensor (i.e. matrix). The element in tensor $\mathcal{T}_{01}$, such as $\mathcal{T}_{01}(2, 1)$, can be calculated from the contraction between $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$ by

$$\mathcal{T}_{01}(2,1) = \sum_{\alpha=0}^{N^0_{\boldsymbol{\tau}_0}} \sum_{\beta=0}^{N^1_{\boldsymbol{\tau}_0}} \boldsymbol{\tau}_0(\alpha, \beta, 2) \times \boldsymbol{\tau}_1(\alpha, \beta, 1). \tag{1}$$

In Figure 2(b), we term $N^2_{\boldsymbol{\tau}_0}$ and $N^2_{\boldsymbol{\tau}_1}$ as free orders which have only one end. These orders will be preserved after the contraction. Furthermore, $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$ share two orders that have the same ends, which are remarked as $N^0_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1}$ and $N^1_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1}$. We call them sharing orders and we have $N^m_{\boldsymbol{\tau}_i\boldsymbol{\tau}_j} = N^m_{\boldsymbol{\tau}_j\boldsymbol{\tau}_i}$. The contraction between two tensors can be interpreted as eliminating sharing orders between two source tensors and preserving all free orders in the new tensor after the contraction.

In this paper, we define the compute expense as how many multiplication operations are required for the contraction. In this example, in order to get one element in $\mathcal{T}_{01}$ we need 8 multiplications (i.e. $N^0_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} \times N^1_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1}$). Hence, 64 multiplications in total (i.e. $N^2_{\boldsymbol{\tau}_0} \times N^0_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} \times N^1_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} \times N^2_{\boldsymbol{\tau}_1}$) are needed to finish the contraction between $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$. Moreover, the new tensor $\mathcal{T}_{01}$ will consume additional storage space, which is defined as the storage expense in this paper. In this example, the storage expense of $\mathcal{T}_{01}$ is 8 (i.e. $N^2_{\boldsymbol{\tau}_0} \times N^2_{\boldsymbol{\tau}_1}$).

In order to simplify the calculation of expense, we adopt the log_k representation of each order in a tensor as suggested

[1]https://github.com/liangling76/tensor-contraction-sequence-searching

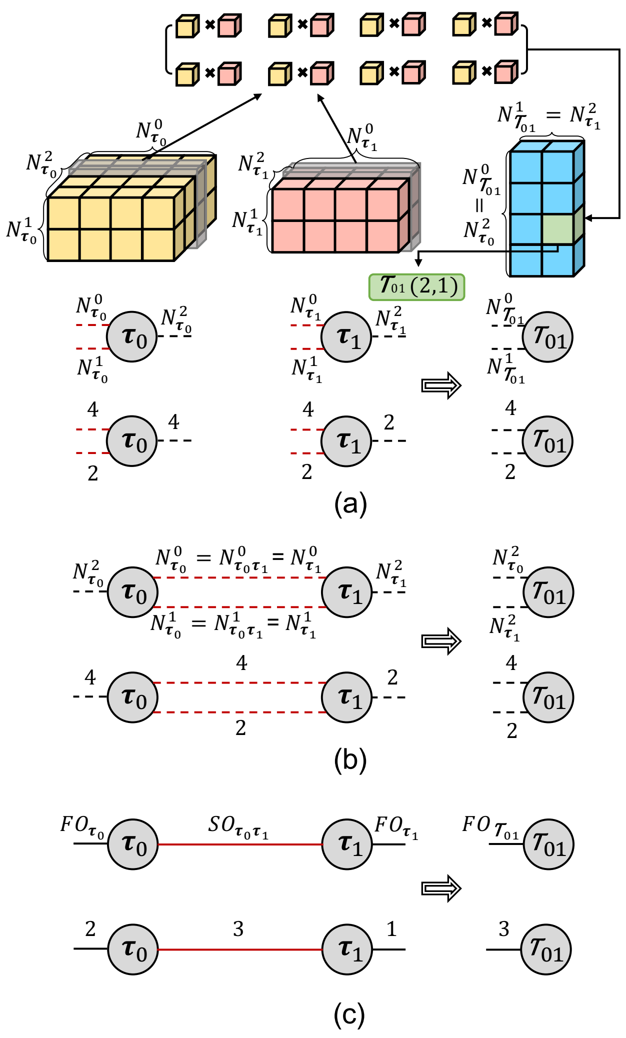


Figure 2: Illustration of the tensor contraction operation between two tensors: (a) Graph representation of two third-order tensors $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$. The free orders include $N^2_{\boldsymbol{\tau}_0}$ and $N^2_{\boldsymbol{\tau}_1}$, the sharing orders are $N^0_{\boldsymbol{\tau}_0}$, $N^0_{\boldsymbol{\tau}_1}$, $N^1_{\boldsymbol{\tau}_0}$, and $N^1_{\boldsymbol{\tau}_1}$; (b) Normal graph representation of the tensor contraction operation; (c) log_k (here $k = 2$) graph representation of the tensor contraction operation.

by [31], where k can be an arbitrary positive value. With the log_k representation, we denote the free and sharing orders with solid lines as Figure 2(c). In this example, we set $k = 2$, then we calculate the log_2 value of each order and denote the free orders as $FO_{\mathcal{T}_I}$. In this example, we now have $FO_{\boldsymbol{\tau}_0} = log_2 N^2_{\boldsymbol{\tau}_0} = 2$ and $FO_{\boldsymbol{\tau}_1} = log_2 N^2_{\boldsymbol{\tau}_1} = 1$. In a similar way, the sharing orders are also collected together and further denoted as $SO_{\mathcal{T}_I\mathcal{T}_J}$ (note that $SO_{\mathcal{T}_I\mathcal{T}_J} = SO_{\mathcal{T}_J\mathcal{T}_I}$). In this example, $SO_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} = log_2(N^0_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} \times N^1_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1}) = log_2 N^0_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} + log_2 N^1_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} = 3$. The log_k representation of collected free and sharing orders are summarized as

$$FO_{\mathcal{T}_I} = \sum_m log_k N^m_{\mathcal{T}_I},\ \{N^m_{\mathcal{T}_I} \in free\ orders\ of\ \mathcal{T}_I\}, \quad (2)$$

$$SO_{\mathcal{T}_I\mathcal{T}_J} = \sum_m log_k N^m_{\mathcal{T}_I\mathcal{T}_J}. \quad (3)$$

Then we use $S_{\mathcal{T}_I}$ to denote the data size of each tensor as

$$S_{\mathcal{T}_I} = FO_{\mathcal{T}_I} + \sum_J SO_{\mathcal{T}_I\mathcal{T}_J}, \quad where\ J \notin I. \quad (4)$$

In this example, we have $S_{\boldsymbol{\tau}_0} = FO_{\boldsymbol{\tau}_0} + SO_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} = 5$. Finally, the calculations of storage expense (SE) and compute expense (CE) for contracting two tensors under the log_k representation are governed by

$$SE_{\mathcal{T}_I\mathcal{T}_J} = S_{\mathcal{T}_{IJ}} = S_{\mathcal{T}_I} + S_{\mathcal{T}_J} - 2SO_{\mathcal{T}_I\mathcal{T}_J}, \quad (5)$$

$$CE_{\mathcal{T}_I\mathcal{T}_J} = S_{\mathcal{T}_I} + S_{\mathcal{T}_J} - SO_{\mathcal{T}_I\mathcal{T}_J} = S_{\mathcal{T}_{IJ}} + SO_{\mathcal{T}_I\mathcal{T}_J}. \quad (6)$$

For instance, we yield $SE_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} = 3$ and $CE_{\boldsymbol{\tau}_0\boldsymbol{\tau}_1} = 6$ in this example. The real storage and compute expenses under this format are $k^{SE_{\mathcal{T}_I\mathcal{T}_J}}$ (i.e. $2^3 = 8$ here) and $k^{CE_{\mathcal{T}_I\mathcal{T}_J}}$ (i.e. $2^6 = 64$ here), respectively. In the rest of this paper, we use the log_k representation unless otherwise specified. From Equation (2)-(6), it is easy to observe that the log_k representation could transform the complex multiplicative calculations to simpler additive ones, which accelerates the processing.

Tensor Network Contraction. In Figure 3(a), we give an example of tensor network with four tensors. In this example, we randomly assign the network topology, the free order of each tensor and sharing orders between tensors. In this tensor network, the original single tensor $\boldsymbol{\tau}_2$ occupies the highest storage space according to Equation (4), i.e. $S_{\boldsymbol{\tau}_2} = FO_{\boldsymbol{\tau}_2} + SO_{\boldsymbol{\tau}_0\boldsymbol{\tau}_2} + SO_{\boldsymbol{\tau}_1\boldsymbol{\tau}_2} = 14$.

For a tensor network with V tensors, $V - 1$ contraction steps in total are required to contract the network into one tensor. We use sq to represent a contraction sequence. We present an arbitrary contraction sequence in Figure 3(b), i.e. $sq = (((\boldsymbol{\tau}_0\boldsymbol{\tau}_1)\boldsymbol{\tau}_2)\boldsymbol{\tau}_3)$. In each contraction step, the highlighted tensors are selected to perform a contraction. In particular, we take the first contraction step to illustrate the contraction operation. In this step, $\boldsymbol{\tau}_0$ and $\boldsymbol{\tau}_1$ are selected for contraction. As mentioned earlier, the contraction first eliminates the sharing orders between these two tensors. Then, two tensors coalesce into one and the sharing orders connected with other tensors will be merged if they have the same ends after contraction. At last, the free orders of these two contracted tensors are also collected together.

Contraction Sequence with the Lowest Maximum Expense. Usually, a contraction sequence can be evaluated by total contraction expense [27], [30], [32] or maximum contraction expense [25], [29], [31]. The total contraction expense is the sum of storage or compute expense at every contraction step; while the maximum contraction expense considers the maximum storage or compute expense across all contraction steps. In essence, the maximum contraction expense becomes close to the total contraction expense when the length of each order is large [31]. Moreover, the maximum contraction expense would be a preferable evaluation metric in practical hardware system. The reason is that for a hardware system, while the off-chip resource is usually sufficient, the on-chip resource for both memory and compute is very limited due to the concern on chip area and fabrication cost. Thus, the algorithm is actually executed step by step with frequent data exchange between off-chip memory and on-chip buffer, rather

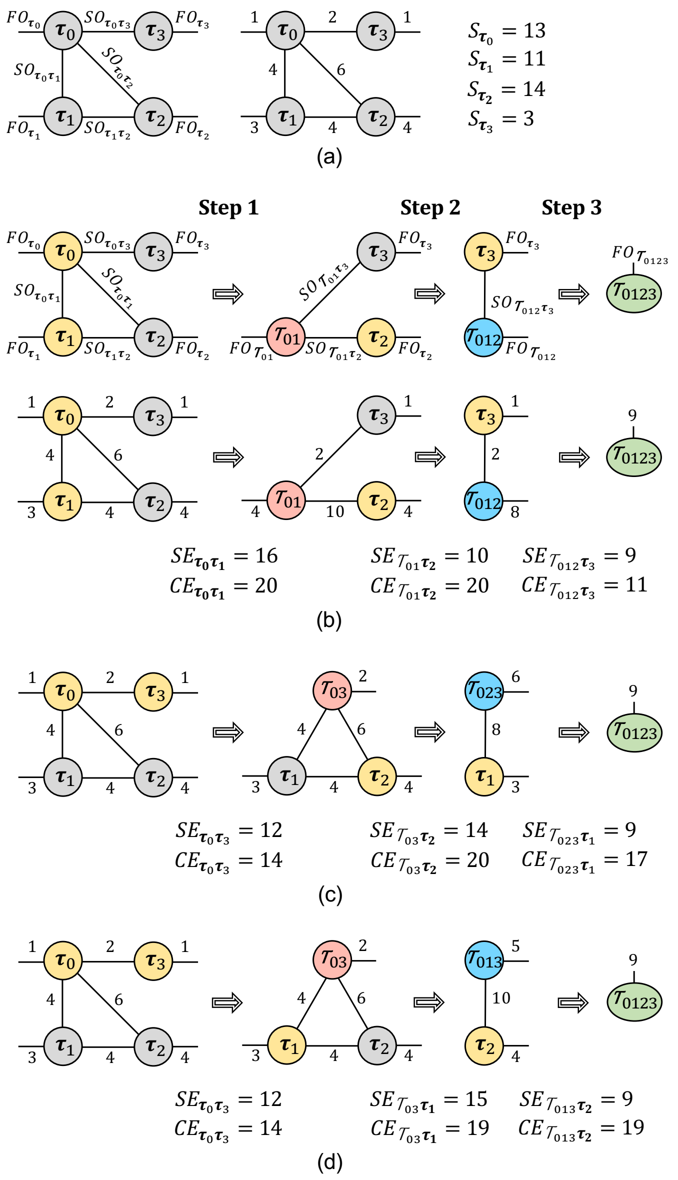


Figure 3: An example of tensor network contraction: (a) A tensor network with randomly assigned topology and order values; (b) An arbitrary contraction sequence; (c) The contraction sequence with the lowest maximum SE; (d) The contraction sequence with the lowest maximum CE.

than hold all the contraction steps on the chip. In this sense, the maximum contraction expense can predict the hardware running performance better. Therefore, in this paper we adopt the maximum SE (MS) and the maximum CE (MC) as the evaluation metrics to measure the contraction cost of a given contraction sequence:

$$MS = max_t\ SE(sq_t),\ MC = max_t\ CE(sq_t) \qquad (7)$$

where t represents the step index and sq_t refers to the t-step tensor contraction in the entire contraction sequence.

In Figure 3(b), we calculate the storage and compute expenses for each contraction step. We find that the first contraction step consumes the highest storage expense to store the tensor $\mathcal{T}_{01}$ after contraction. The first two steps consume the highest compute expense to calculate tensors $\mathcal{T}_{01}$ and $\mathcal{T}_{012}$. Intuitively, different contraction sequences have different MS and MC values. The goal of this work is to find a contraction sequence that has the lowest MS or MC. Figure 3(c) shows an expected optimal contraction sequence with the lowest MS, i.e. $sq = (((\boldsymbol{\tau}_0\boldsymbol{\tau}_3)\boldsymbol{\tau}_2)\boldsymbol{\tau}_1)$. Both the MS and the MC occur in the second contraction step. The contraction sequence with the lowest MC is given in Figure 3(d), i.e. $sq = (((\boldsymbol{\tau}_0\boldsymbol{\tau}_3)\boldsymbol{\tau}_1)\boldsymbol{\tau}_2)$. The MS also occurs in the second contraction step; while the MC occurs in the last two steps. From the instances, it is easy to observe that the contraction sequence with the lowest MS (or MC) cannot guarantee that the sequence has the lowest MC (or MS). Thus, the metric should be selected in advance to evaluate the contraction cost in different scenarios.

B. BFS Search Algorithm

In this paper, we adopt the BFS algorithm as the design basis [26], [27], which is illustrated in Figure 4. In this example, the network includes three tensors and we need two contraction steps to perform the network contraction. We use Set_v to denote the set of possible tensors which are contracted by v original tensors and have v subscript numbers. Each Set_v contains C_V^v possible tensors, e.g. $Set_2 = \{\mathcal{T}_{01}, \mathcal{T}_{02}, \mathcal{T}_{12}\}$. For each possible tensor, we can find $Split_v$ split cases to divide its v subscript numbers into two sets representing two split source tensors which can produce that possible tensor in one contraction step. The search space of each $split_v$ is $O(2^v)$. There are three possible tensors in Set_2 (i.e. $\mathcal{T}_{01}$, $\mathcal{T}_{02}$, and $\mathcal{T}_{12}$), each of which has one split case with two split source tensors that can be found in Set_1. In Set_3, the unique possible tensor $\mathcal{T}_{012}$ has three split cases, i.e. $Split_3 = 3$, and $\mathcal{T}_{012}$ can be contracted through $\{\boldsymbol{\tau}_0, \mathcal{T}_{12}\}$, $\{\boldsymbol{\tau}_1, \mathcal{T}_{02}\}$, or $\{\boldsymbol{\tau}_2, \mathcal{T}_{01}\}$ with a contraction operation between one split source tensor in Set_1 and the other in Set_2. The number of split cases for each possible tensor in Set_v is

$$Split_v = \begin{cases} \sum_{k=1}^{\lfloor v/2 \rfloor} C_V^v, & \text{if v is odd} \\ \sum_{k=1}^{v/2-1} C_V^v + \frac{C_V^{v/2}}{2}, & \text{if v is even} \end{cases} . \qquad (8)$$

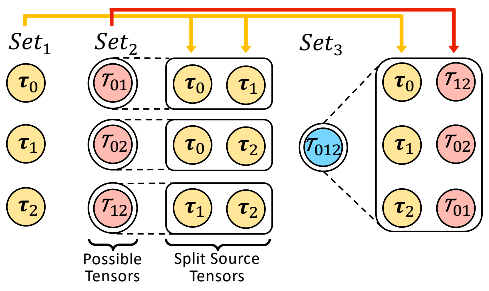


Figure 4: Illustration of the BFS algorithm in a network with three tensors. The tensors surrounded with a big circle are termed as "possible tensors", and the two tensors on the same row in a split case that can produce each possible tensor are termed as "split source tensors".

In the BFS algorithm, we calculate and save the lowest MS (or MC) for producing every possible tensor and the corresponding split case (the search starts from Set_2 to Set_V).

Finally, we only need to consider the contraction expense of the last search iteration. The reason is that, in the last iteration, the lowest MS (or MC) of producing all split source tensors can be found in the previous iterations, which have already been calculated and saved earlier. Thus, after going through all split cases in the last iteration, the lowest MS (or MC) to produce each final possible tensor and the corresponding split case can be acquired. Based on that we are able to determine the optimal sequence through the reverse trace of the lowest MS (or MC). The detail of our algorithm implementation will be introduced in Section III-B.

III. Adjacency Matrix-based Contraction Search

In this section, we first describe how to use the adjacency matrix to update the tensor network during contraction and how to apply the outer product pruning based on the adjacency matrix data structure in Section III-A. Then, we introduce the algorithm details for both the vanilla BFS search and the improved search with outer product pruning in Section III-B. Finally, we present the parallelism optimization to further improve the performance in Section III-C.

A. Data Structure Design

From the above analysis, the only computation during the search is to calculate SE or CE for each split case of possible tensors at each iteration according to Equation (5)-(6). Thanks to the log_k representation, these calculations only include additive operations. However, we still need to get the data size of two split source tensors in each split case and the sharing order between them. For faster processing, we propose an efficient data structure based on adjacency matrix.

A tensor network can be viewed as an undirected graph. The adjacency matrix format of a network with four tensors is shown in Figure 5(a). Note that we do not include free orders in the adjacency matrix. In general, we define each element in the matrix to satisfy $E_{\boldsymbol{\tau}_i\boldsymbol{\tau}_j} + E_{\boldsymbol{\tau}_j\boldsymbol{\tau}_i} = SO_{\boldsymbol{\tau}_i\boldsymbol{\tau}_j} = SO_{\boldsymbol{\tau}_j\boldsymbol{\tau}_i}$ and $E_{\boldsymbol{\tau}_i\boldsymbol{\tau}_i} = 0$. Any configuration with non-negative elements that satisfy the above equation is allowed. The extreme configuration of $E_{\boldsymbol{\tau}_i\boldsymbol{\tau}_j}$ (or $E_{\boldsymbol{\tau}_j\boldsymbol{\tau}_i}$) $= SO_{\boldsymbol{\tau}_i\boldsymbol{\tau}_j} = SO_{\boldsymbol{\tau}_j\boldsymbol{\tau}_i}$ and $E_{\boldsymbol{\tau}_j\boldsymbol{\tau}_i}$ (or $E_{\boldsymbol{\tau}_i\boldsymbol{\tau}_j}$) $= 0$ is also acceptable.

Sharing Order and Data Size. For a given split case of a possible tensor at each iteration, the sharing order between two split source tensors is the summation of all orders that connect these two tensors as Equation (3). Assume we want to find the sharing order between two split source tensors $\boldsymbol{\tau}_1$ and $\mathcal{T}_{02}$, we have $SO_{\boldsymbol{\tau}_1\mathcal{T}_{02}} = SO_{\boldsymbol{\tau}_1\boldsymbol{\tau}_0} + SO_{\boldsymbol{\tau}_1\boldsymbol{\tau}_2}$. Figure 5(a) boxes all elements which are required to compute the sharing order between them. If we directly use the above naive calculation, the data access to the adjacency matrix according to the tensor subscripts usually results in large cost due to the random access pattern in a 2D space and non-reusable additions. To address this issue, in Figure 5(b), we design a more efficient way to calculate the sharing order which requires two steps: (1) For each split source tensor, all of its involved original rows in the adjacency matrix are reduced into a new row vector $R_{\mathcal{T}_I}$ via the accumulation operation; (2) For each row vector of the two split source tensors, the elements whose tensor subscription appears in the other split source tensor will be selected and added to produce the sharing order. Specifically, we first get the row vector $R_{\mathcal{T}_{02}}$ by accumulating the involved original rows in the adjacency matrix and $R_{\boldsymbol{\tau}_1}$ from the original adjacency matrix. Then, the sharing order can be acquired by $SO_{\mathcal{T}_{02}\boldsymbol{\tau}_1} = R_{\mathcal{T}_{02}}[1] + R_{\boldsymbol{\tau}_1}[0] + R_{\boldsymbol{\tau}_1}[2]$. We use the tensor network in Figure 3 as an example to illustrate the sharing order calculation process. The adjacency matrix of the sharing orders in the original tensor network is shown in Figure 5(c). We first get the row vector $R_{\mathcal{T}_{02}}$ by summing row vectors $R_{\boldsymbol{\tau}0}$ and $R_{\boldsymbol{\tau}_2}$ as in Figure 5(d). Then, the sharing order $SO_{\mathcal{T}_{02}\boldsymbol{\tau}_1}$ can be calculated by accumulating the yellow box in $R_{\mathcal{T}_{02}}$ and the red boxes in $R_{\boldsymbol{\tau}_1}$, which equals 8.

It is worthy noting that the saved row vectors (e.g. $\mathcal{T}_{02}$) will be frequently reused during the entire search process towards the optimal contraction sequence, which significantly reduces the access and calculation costs compared to the naive scheme solely based on the original adjacency matrix. As the search iteration goes on, the required new row vectors can be efficiently obtained given the previously saved row vectors:

$$R_{\mathcal{T}_I} = R_{\mathcal{T}_J} + R_{\mathcal{T}_K}, \; J \cup K = I \; \wedge \; J \cap K = \varnothing, \tag{9}$$

which avoids redundant accesses to the original adjacency matrix. Finally we save the row vectors for all possible tensors. Furthermore, the data size of each possible tensor can be obtained by calculating the SE of an arbitrary split case as Equation (5), which is also saved for reuse in the calculation of consequent SEs and CEs during the search.

Outer Product Pruning. The contraction of two tensors without any sharing order is called an outer product, e.g. the contraction between $\boldsymbol{\tau}_2$ and $\boldsymbol{\tau}_3$ in the tensor network provided in Figure 3. In prior work, it has been demonstrated, for an arbitrary tensor network, there always exists a contraction sequence which achieves the lowest MS or MC and does not include any possible tensor that has split cases with outer product [31]. Hence, the possible tensors with outer-product split cases can be pruned during the search of the optimal contraction sequence. The related calculation of these possible tensors can be removed accordingly. When the tensor network is composed by several sub-networks which do not share orders with each other, the contraction can be done by first applying tensor contraction in each sub-network and then do outer product between them with an arbitrary contraction sequence. To judge whether a possible tensor has outer-product split cases can also be solved by BFS or minimum cut [33], whereas, they are not efficient. Especially, when a tensor network has dense connections, the benefits gained from search space reduction will be degraded. To this end, we design a fast method to identify all possible tensors that can be pruned, which is still based on the adjacency matrix data structure.

At the beginning, an additional adjacency matrix is generated, as depicted in Figure 5(e). Compared to the original adjacency matrix, $E_{\boldsymbol{\tau}_{ii}}$ is set to 1. For a given possible tensor $\mathcal{T}_I$, we aim to find out all original single tensors $\boldsymbol{\tau}_j$ that do not share orders with $\mathcal{T}_I$. For example, we want to figure out all $\boldsymbol{\tau}_j$ that disconnect to $\mathcal{T}_{12}$. According to Equation (3), $\mathcal{T}_I$ and $\boldsymbol{\tau}_j$ are disconnected when $SO_{\mathcal{T}_I\boldsymbol{\tau}_j} = 0$. In the adjacency

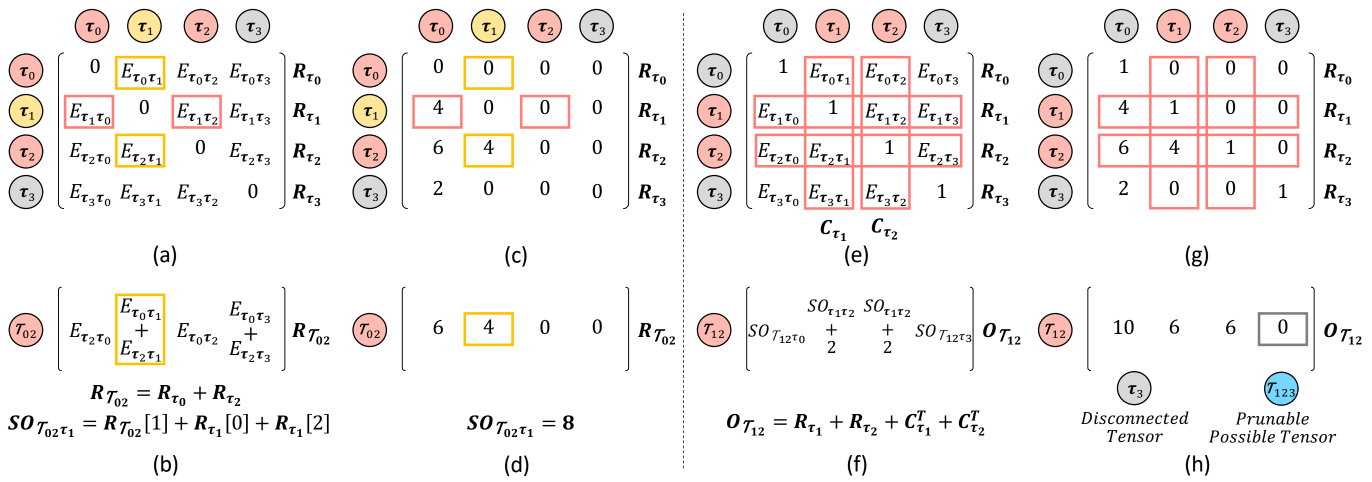


Figure 5: Overview of the adjacency matrix based contraction sequence search: (a) Original adjacency matrix for sharing orders of a network with four tensors; (b) Sharing order calculation based on row vectors (R); (c) Original adjacency matrix for sharing orders of the tensor network in Figure 3; (d) Example of the sharing order calculation; (e) Adjacency matrix used in the outer product pruning; (f) Outer product vector calculation; (g) Adjacency matrix used in the outer product pruning of the tensor network in Figure 3; (h) Example of the outer product vector calculation, and the corresponding disconnected tensor and pruned possible tensor.

matrix, all the information about sharing orders of $\mathcal{T}_I$ are stored in the involved rows and columns (i.e. #1 and #2 rows, #1 and #2 columns here in the red boxes of Figure 5(e)). Based on the sharing order calculation principles, the outer product vector $O_{\mathcal{T}_I}$ can be obtained by accumulating the boxed row and column vectors, as illustrated in Figure 5(f). From the outer product vector $O_{\mathcal{T}_{12}}$, it is easy to observe that the first element indicates the sharing order between $\mathcal{T}_{12}$ and $\boldsymbol{\tau}_0$, and the last element represents the sharing order between $\mathcal{T}_{12}$ and $\boldsymbol{\tau}_3$. The location of zero elements in $O_{\mathcal{T}_{12}}$ can reflect which original single tensors are disconnected to $\mathcal{T}_{12}$. Note that the single tensors whose subscript is already in I itself should not be selected. This is the reason that we set $E_{\boldsymbol{\tau}_{ii}}$ to 1, resulting the elements' value greater than 0 in $O_{\mathcal{T}_I}$ when $i \in I$ (i.e. $O_{\mathcal{T}_{12}}[1]$ and $O_{\mathcal{T}_{12}}[2]$ here). Then, based on the locations of zero elements, we can infer the prunable possible tensors which can be contracted by two disconnected tensors and one of them is $\mathcal{T}_{12}$.

For the example tensor network in Figure 3, the corresponding outer product adjacency matrix is shown in Figure 5(g). For the possible tensor $\mathcal{T}_{12}$, we find that $\boldsymbol{\tau}_3$ is the original disconnected tensor, because the last element in the outer product vector $O_{\mathcal{T}_{12}}$ is zero as in Figure 5(h). Then, we can conclude that the possible tensor $\mathcal{T}_{123}$ can be pruned from the search space. Since it can be produced by two split tensors which do not share orders and one of the split tensor is $\mathcal{T}_{12}$.

B. Contraction Sequence Search Algorithm

Adjacency Matrix based Vanilla Search. Based on the efficient calculation of sharing orders and prunable possible tensors shown in Figure 5, we further detail the algorithm of vanilla BFS search with the proposed adjacency matrix based data structure. Before going through the search algorithm, we design a function named SOC to calculate the sharing orders between two split source tensors $\mathcal{T}_{I_1}$ and $\mathcal{T}_{I_2}$. The inputs of this function are row vectors ($R_{\mathcal{T}_{I_1}}$ and $R_{\mathcal{T}_{I_2}}$) and tensor subscripts (I_1 and I_2) of two split source tensors. The implementation is given in Algorithm 1.

Algorithm 1: Sharing Order Calculation (SOC)

```
1 Function SOC($R_{\mathcal{T}_{I_1}}$, $R_{\mathcal{T}_{I_2}}$, $I_1$, $I_2$)
2     $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} = 0$;
3     for all $i \in I_1$ do $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} += R_{\mathcal{T}_{I_2}}[i]$; end
4     for all $i \in I_2$ do $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} += R_{\mathcal{T}_{I_1}}[i]$; end
5     return $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}}$
6 end
```

The overall search algorithm is provided in Algorithm 2. Notice that LME stores the lowest maximum expense required to produce all possible tensors. At the beginning, the data size $S_{\boldsymbol{\tau}_i}$, the row vector $R_{\boldsymbol{\tau}_i}$, and the $LME_{\boldsymbol{\tau}_i}$ for each single tensor in Set_1 are initialized. The initialization of LME depends on different evaluation metrics (MS or MC).

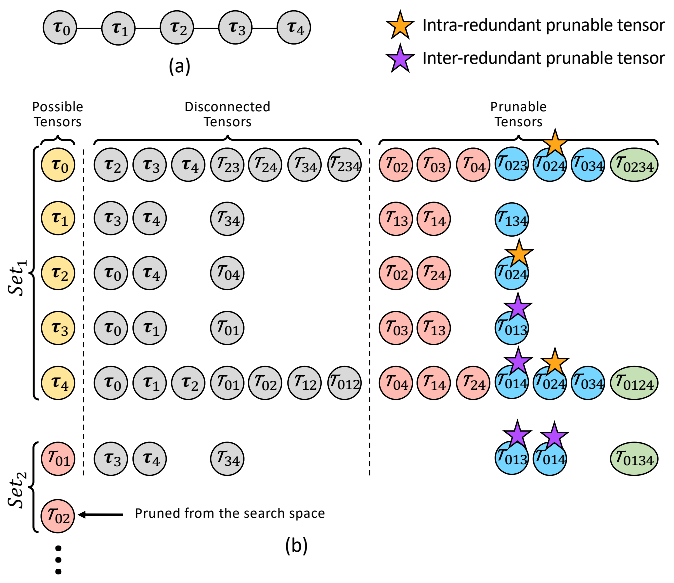
During the search, the first outer loop goes through all sets and the second outer loop traverses all possible tensors of the selected set. At the beginning of the second outer loop, we will calculate the size of the current possible tensor (i.e. $S_{\mathcal{T}_I}$) and its row vector representation (i.e. $R_{\mathcal{T}_I}$) based on an arbitrary split case according to Equation (4) and (9), respectively. If the measurement metric is MS, the SE of $\mathcal{T}_I$ will be saved as the contraction expense (i.e. $expense$) of $\mathcal{T}_I$. The inmost loop traverses all split cases of $\mathcal{T}_I$. For each split case, if the measurement metric is MC, the CE of contracting two split source tensors is calculated and updated into $expense$. Then, the maximum expense required for producing $\mathcal{T}_I$ under the current split case (i.e. ME_{tmp}) can be obtained by taking the maximum value from $expense$ and the LMEs of producing two split tensors, where the latter values have been calculated and saved in the previous iteration. If ME_{tmp} is smaller than the LME value saved in $LME_{\mathcal{T}_I}$, $LME_{\mathcal{T}_I}$ will be

Algorithm 2: Adjacency Matrix based Vanilla Search

```
Initialize the adjacency matrix $adj_{mat}$;
Initialize all elements in $LME$ with $MAX\_Float$;
for $v = 0 : V - 1$ do
    $S_{\boldsymbol{\tau}_v} = sum(adj_{mat}[v, :]) + sum(adj_{mat}[:, v]) + FO_{\boldsymbol{\tau}_v}$;
    $R_{\boldsymbol{\tau}_v} = adj_{mat}[v, :]$;
    if to find the lowest MS then $LME_{\boldsymbol{\tau}_v} = S_{\boldsymbol{\tau}_v}$; end
    else if to find the lowest MC then $LME_{\boldsymbol{\tau}_v} = 0$; end
end
//Traverse all sets
for $v = 2 : V$ do
    //Traverse all possible tensors in each set
    for all $\mathcal{T}_I \in Set_v$ do
        Select an arbitrary split case of $\mathcal{T}_I$ with $\mathcal{T}_{I_1}$ and $\mathcal{T}_{I_2}$;
        $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} = SOC(R_{I_1}, R_{I_2}, I_1, I_2)$;
        $S_{\mathcal{T}_I} = S_{\mathcal{T}_{I_1}} + S_{\mathcal{T}_{I_2}} - 2SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}}$;
        $R_{\mathcal{T}_I} = R_{\mathcal{T}_{I_1}} + R_{\mathcal{T}_{I_2}}$;
        if to find the lowest MS then
            $expense = S_{\mathcal{T}_I}$;
        end
        //Traverse all split cases
        for all split cases of $\mathcal{T}_I$ do
            Get two split source tensors $\mathcal{T}_{I_1}$ and $\mathcal{T}_{I_2}$;
            if to find the lowest MC then
                $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} = SOC(R_{\mathcal{T}_{I_1}}, R_{\mathcal{T}_{I_2}}, I_1, I_2)$;
                $expense = S_{\mathcal{T}_I} + SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}}$;
            end
            $ME_{tmp} = max(expense, LME_{\mathcal{T}_{I_1}}, LME_{\mathcal{T}_{I_2}})$;
            if $ME_{tmp} < LME_{\mathcal{T}_I}$ then
                $LME_{\mathcal{T}_I} = ME_{tmp}$;
                $sq_{\mathcal{T}_I} = [I_1, I_2]$;
            end
            if to find the lowest MS and $ME_{tmp} == expense$
              then
                break;
            end
        end
    end
end
```

updated with ME_{tmp}, and the optimal split case for $\mathcal{T}_I$ will also be updated into sq. If the measurement metric is MS, we should note although the saved LMEs of producing two split tensors are variable across split cases, the SE value of $\mathcal{T}_I$ does not change. Therefore, if the largest SE occurs in the last contraction step (i.e. ME_{tmp} equals $expense$), there is no need to search the rest split cases since the ME_{tmp} value cannot be smaller than current $LME_{\mathcal{T}_I}$ anymore and no update of $LME_{\mathcal{T}_I}$ will occur.

Improved Search with Outer Product Pruning. In the previous subsection, we have illustrated how to get disconnected tensors and prunable tensors based on the outer product vector $O_{\mathcal{T}_I}$ of a possible tensor $\mathcal{T}_I$. Here, we further detail how we apply the outer product pruning to the search algorithm. We explain it using a chain structure tensor network with five tensors as depicted in Figure 6(a). The search space pruning based on outer product is shown in Figure 6(b). For the possible tensors in Set_1, we list all disconnected tensors and prunable possible tensors for each original single tensor. In Set_2, we need to get the LME and perform the outer product search for the first possible tensor $\mathcal{T}_{01}$, because it is not excluded from the search space in Set_1. For the prunable possible tensor $\mathcal{T}_{02}$, we do not search its optimal contraction sequence, and also we skip its outer product search. Later we will demonstrate that our search strategy is able to find all prunable possible tensors.



$\boldsymbol{\tau}_0$ $\boldsymbol{\tau}_1$ $\boldsymbol{\tau}_2$ $\boldsymbol{\tau}_3$ $\boldsymbol{\tau}_4$
(a)
Intra-redundant prunable tensor
Inter-redundant prunable tensor
Possible Tensors
Disconnected Tensors
Prunable Tensors
Set_1
Set_2
Pruned from the search space
(b)

Figure 6: Outer product pruning during the search process: (a) An example tensor network with a chain structure; (b) The outer product pruning for all original tensors in Set_1 and two possible tensors in Set_2.

When we apply the outer product pruning, we find that the prunable possible tensors might be recognized multiple times. For the prunable possible tensors identified by the possible tensors from the same set, we call them intra-redundant prunable tensors (e.g. $\mathcal{T}_{024}$ marked by orange stars). For those prunable possible tensors identified by the possible tensors from different sets, we call them inter-redundant prunable tensors (e.g. $\mathcal{T}_{013}$ and $\mathcal{T}_{014}$ marked by purple stars). In this work, we propose a method to reduce the recognition times for inter-redundant prunable tensors, which is presented in Algorithm 3.

Algorithm 3: Outer Product Pruning (OPP)

```
Function OPP($O_{\mathcal{T}_I}$, $v_I$)
    Find set $J = \{j \mid O_{\mathcal{T}_I}[j] = 0\}$;
    for all $J' \subseteq J$ & $v_{J'} \geqslant v_I$ do
        $P_{\mathcal{T}_{J' \cup I}} = 0$; //Pruned from the search space.
    end
end
```

We use $P_{\mathcal{T}_I}$ to indicate whether a possible tensor will be pruned from the search space (0 pruned, 1 preserved). The input of the function OPP (outer product pruning) is the outer product vector $O_{\mathcal{T}_I}$ and the number of elements in the subscript I which is denoted as v_I (v_I represents how many original single tensors are involved in producing $\mathcal{T}_I$). In the function OPP, we find all prunable possible tensors based on the possible tensor $\mathcal{T}_I$. The possible tensors $\mathcal{T}_{J' \cup I}$ can be pruned from the search space, where $\mathcal{T}_{J'}$ represents the possible tensors disconnected to $\mathcal{T}_I$. Note that during our pruning, we only consider the disconnected tensors $\mathcal{T}_{J'}$ which satisfy $v_{J'} \geqslant v_I$. The reason is that during the previous search of $\mathcal{T}_{J''}$, when $v_{J''} < v_I$, the possible tensor $\mathcal{T}_{J'' \cup I}$ has already been pruned. With our method, parts of inter-redundant pruning can be reduced. For example, in Figure 6(b), when we apply OPP to $\mathcal{T}_{01}$, the disconnected tensors do not include $\boldsymbol{\tau}_3$ and $\boldsymbol{\tau}_4$ because $\mathcal{T}_{013}$ and $\mathcal{T}_{014}$ have already been pruned. In this way, we prevent the redundant recognition of the pruned possible tensors $\mathcal{T}_{013}$ and $\mathcal{T}_{014}$.

We first give a sketch proof that our pruning method is sufficient to find all prunable possible tensors. For an arbitrary prunable possible tensor $\mathcal{T}_I$ which can be contracted by a set of possible tensors $T = \{\mathcal{T}_{I_i} \mid i = 0, 1, ..., m\}$, where $I_j \cap I_k = \phi$ and $SO_{\mathcal{T}_{I_j}\mathcal{T}_{I_k}} = 0$ if $j \neq k$ (note that all $\mathcal{T}_{I_i}$ are not prunable possible tensors). Suppose $v_{I_j} \leqslant v_{I_i}$, for $i = 0, 1, ..., m$. Obviously, $v_{I_j} \leqslant v_I/2 \leqslant v_{I \setminus I_j}$. Therefore, each prunable possible tensor $\mathcal{T}_I$ must be pruned during the pruning stage of $\mathcal{T}_{I_j}$. Our outer product pruning algorithm will go through all possible tensors which can be contracted without outer product and begin to search and remove the prunable possible tensors from Set_1. Therefore, it is sufficient to find all prunable possible tensors. Next, we analyze the complexity of finding a prunable possible tensor (i.e. the recognition times for a prunable possible tensor). Based on our pruning process, for an arbitrary prunable possible tensor described previously, it will be found from the subset T' of T, where $T' = \{\mathcal{T}_{I_j} \mid v_{I_j} \leqslant v_{I_k},\ j, k = 0, 1, ..., m\}$. Apparently, the number of elements in T' is smaller than or equals the number of elements in the subscript I. Thus, the complexity is v_I.

The overall algorithm is shown in Algorithm 4. At the initialization stage for each original tensor in Set_1, besides the data size $S_{\boldsymbol{\tau}_v}$, the row vector $R_{\boldsymbol{\tau}_v}$, and the lowest maximum expense $LME_{\boldsymbol{\tau}_v}$, the outer product vector $O_{\boldsymbol{\tau}_v}$ is also calculated and the first round of the outer product pruning is then executed. During the search, for each iteration in the second outer loop, only those possible tensors whose $P_{\mathcal{T}_I}$ value equals one will be selected to run the further contraction expense counting. Also, for each split case of $\mathcal{T}_I$, if one of the split source tensor (i.e. $\mathcal{T}_{I_1}$ or $\mathcal{T}_{I_2}$) has been marked to be pruned, the current split case is bypassed since it will not appear in the final optimal contraction sequence. In contrast to the vanilla search algorithm, we need to update the outer product vector $O_{\mathcal{T}_I}$ for each possible tensor when we apply the outer product pruning during search.

C. Parallelism Optimization

One advantage of our proposed algorithms is that all of them are easy to execute in parallel. When we calculate the contraction expense of possible tensors in a Set_v, the processing of each one is independent from others. Therefore, there are opportunities for parallelism optimization. Figure 7 shows an example of two-thread parallel optimization when calculating the contraction expense of possible tensors in Set_3, in the case of a network with four tensors.

In our vanilla search (without outer product pruning), for each possible tensor we calculate the storage expense, row

Algorithm 4: Improved Search with Outer Product Pruning

```
Initialize the adjacency matrix $adj_{mat}$;
Initialize all elements in $LME$ with $MAX\_Float$;
Copy $adj_{mat}$ to $adj_{mat_O}$ and set diagonal elements to 1
Initialize all elements in $P$ with 1;
for $v = 0 : V - 1$ do
    $S_{\boldsymbol{\tau}_v} = sum(adj_{mat}[v,:]) + sum(adj_{mat}[:,v]) + FO_{\boldsymbol{\tau}_v}$;
    $R_{\boldsymbol{\tau}_v} = adj_{mat}[v,:]$;
    $O_{\boldsymbol{\tau}_v} = adj_{mat_O}[v,:] + adj_{mat_O}[:,v]^T$;
    if to find the lowest MS then $LME_{\boldsymbol{\tau}_v} = S_{\boldsymbol{\tau}_v}$; end
    else if to find the lowest MC then $LME_{\boldsymbol{\tau}_v} = 0$; end
    $OPP(O_{\boldsymbol{\tau}_v}, 1)$; //Apply outer product pruning to $\boldsymbol{\tau}_v$
end
//Traverse all sets
for $v = 2 : V$ do
    //Traverse all possible in each set
    for all $\mathcal{T}_I \in Set_v$ and $P_{\mathcal{T}_I} = 1$ do
        $flag = False$;
        //Traverse all split cases
        for all split cases of $\mathcal{T}_I$ do
            Get two split source tensors $\mathcal{T}_{I_1}$ and $\mathcal{T}_{I_2}$;
            if $P_{\mathcal{T}_{I_1}} = 0$ or $P_{\mathcal{T}_{I_2}} = 0$ then Continue; end
            if $flag = False$ then
                $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} = SOC(R_{\mathcal{T}_{I_1}}, R_{\mathcal{T}_{I_2}}, I_1, I_2)$;
                $S_{\mathcal{T}_I} = S_{\mathcal{T}_{I_1}} + S_{\mathcal{T}_{I_2}} - 2SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}}$;
                $R_{\mathcal{T}_I} = R_{\mathcal{T}_{I_1}} + R_{\mathcal{T}_{I_2}}$;
                $O_{\mathcal{T}_I} = O_{\mathcal{T}_{I_1}} + O_{\mathcal{T}_{I_2}}$;
                if to find the lowest MS then
                    $expense = S_{\mathcal{T}_I}$;
                end
                $flag = True$;
            end
            if to find the lowest MC then
                $SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}} = SOC(R_{\mathcal{T}_{I_1}}, R_{\mathcal{T}_{I_2}}, I_1, I_2)$;
                $expense = S_{\mathcal{T}_I} + SO_{\mathcal{T}_{I_1}\mathcal{T}_{I_2}}$;
            end
            $ME_{tmp} = max(expense, LME_{\mathcal{T}_{I_1}}, LME_{\mathcal{T}_{I_2}})$;
            if $ME_{tmp} < LME_{\mathcal{T}_I}$ then
                $LME_{\mathcal{T}_I} = ME_{tmp}$;
                $sq = [I_1, I_2]$;
            end
            if to find the lowest MS and $ME_{tmp} == expense$
              then
                break;
            end
        end
        $OPP(O_{\mathcal{T}_I}, v_I)$; //Apply outer product pruning to $T_I$
    end
end
```

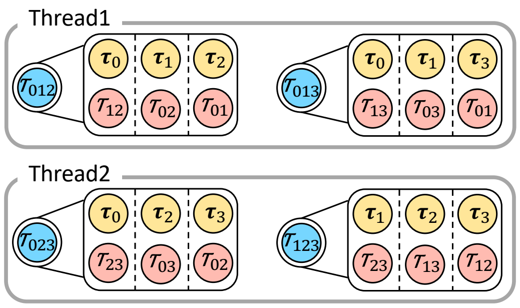

Thread1
$\mathcal{T}_{012}$
$\boldsymbol{\tau}_0$ $\boldsymbol{\tau}_1$ $\boldsymbol{\tau}_2$
$\mathcal{T}_{12}$ $\mathcal{T}_{02}$ $\mathcal{T}_{01}$
$\mathcal{T}_{013}$
$\boldsymbol{\tau}_0$ $\boldsymbol{\tau}_1$ $\boldsymbol{\tau}_3$
$\mathcal{T}_{13}$ $\mathcal{T}_{03}$ $\mathcal{T}_{01}$
Thread2
$\mathcal{T}_{023}$
$\boldsymbol{\tau}_0$ $\boldsymbol{\tau}_2$ $\boldsymbol{\tau}_3$
$\mathcal{T}_{23}$ $\mathcal{T}_{03}$ $\mathcal{T}_{02}$
$\mathcal{T}_{123}$
$\boldsymbol{\tau}_1$ $\boldsymbol{\tau}_2$ $\boldsymbol{\tau}_3$
$\mathcal{T}_{23}$ $\mathcal{T}_{13}$ $\mathcal{T}_{12}$

Figure 7: Illustration of the multithread optimization.

vectors, and update the LME by traversing all split cases. Apparently, these operations do not have memory access conflicts between the possible tensors in the same set. When we apply the outer product pruning in the search process, one additional operation is to update the outer product vector. Fortunately, each possible tensor only updates its own outer product vector. Another operation during search space reduction is to determine whether a possible tensor will be excluded out from final optimal contraction sequence or not. Although different possible tensors may write the same $P_{\mathcal{T}_I}$ simultaneously, the values of them are identical without incurring incorrect results.

IV. Experimental Results

A. Experiment Setup

Most of our experiments were conducted on an Intel Core i7 processor (2.8 GHz) with 16 GB DDR3 DRAM (1600 MHz), similar with [27] for fair comparison. Moreover, to validate the parallelism optimization in Section IV-D, we further tested on an Intel Xeon processor (2.5GHz) with 384GB DDR4 RAM (2133MHz) for the support of more threads. We evaluate on tensor networks with three basic topologies: chain, binary tree, and radial network, as presented in Figure 9. Since the basic topology is not sophisticated enough, we further injected random edges. In short, our network construction includes two stages: (1) initializing a connective network with $V-1$ edges based on one of the basic topologies; (2) adding edges with random locations into the network. All the FO and SO values are positive. In our experiments, we set all SO with the same value which is five times larger than all FO values.

B. Vanilla Search without Space Reduction

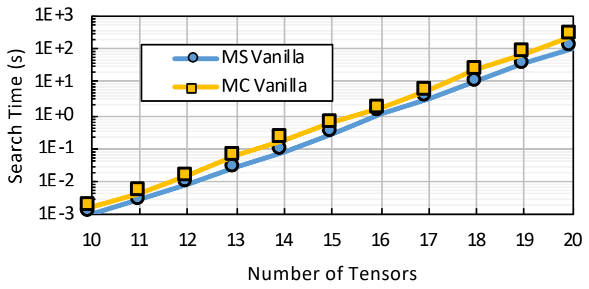


Figure 8: Search time under different measurement metrics without search space reduction.

Table II: Search time (s) under different network scale (i.e. the number of tensors) and measurement metrics.

Net. Scale	15	16	17	18	19	20
MS Vanilla	0.30	1.17	3.25	10.61	35.85	105.89
MC Vanilla	0.58	1.71	5.80	23.44	70.54	254.43

In this subsection, the tensor networks we evaluate are dense networks, i.e. fully connected networks. In the vanilla BFS search without space reduction, we apply exhaustive search, which means that the LME of all possible tensors will be searched. Figure 8 shows the result of search time under different measurement metrics as the number of tensors in the network increases. Table II provides detailed time data in the cases of more than 15 tensors.

The first observation is that finding the lowest MS is cheaper than finding the lowest MC. One reason is that in the calculation of CE, we need the sharing order between split source tensors in all split cases; however, in the calculation of SE we only need the sharing order once from an arbitrary split case. Another reason is due to the proposed early stop mechanism in searching the lowest MS (see line 32-34 in Algorithm 2). The second observation is that when the number of tensors in the network increases by one, the search time grows about three times. This increment relies on the number of split cases during the entire search, which is

$$Split_{total} = \sum_{v=1}^{V} C_V^v \cdot Split_v = \sum_{v=1}^{V} C_V^v \cdot O(2^v) = O(3^V) \quad (10)$$

where we assume the network contains V tensors. It can be seen that the search space is $O(3^V)$.

C. Improved Search with Space Reduction

In this subsection, we evaluate the performance benefited from the search space reduction via outer product pruning. As aforementioned in Section IV-A, we test on three kinds of tensor networks with different portion of extra edges. In order to guarantee the reproducibility and fairness across different network topologies, we adopt the same random seed in all experiments.

Table III: Search time (s) with and without outer product pruning under different evaluation metrics and basic topologies. Here each tensor network contain 19 tensors.

	MS Vanilla	MS Pruning	MC Vanilla	MC Pruning
Chain	0.83	0.27	70.54	0.30
Tree	2.45	1.28	70.54	2.03
Radial	28.70	26.63	70.54	27.50

We first investigate the performance gain from applying the outer product pruning for tensor networks with different basic topologies. Here each tensor network contains 19 tensors and does not include extra edges. The comparison is shown in Table III. Apparently, our pruning strategy can reduce the search time for all networks under both evaluations metrics when compare to the vanilla search. Another observation is that for the vanilla search, when the evaluation metric is MS, the search time significantly varies across different network topologies. The reason is caused by the variance of the early stop mechanism that has variable effects on different network topologies.

Also, with our pruning strategy, the search time similarly varies across different network structures under both metrics. The detailed analysis is presented in Figure 9, which provides the search time with the outer product pruning on networks under different basic topologies and tensor numbers. From the results, we can get several observations. First, with the same reduction strategy, finding the lowest MS is still faster than finding the lowest MC, which is similar with the observation

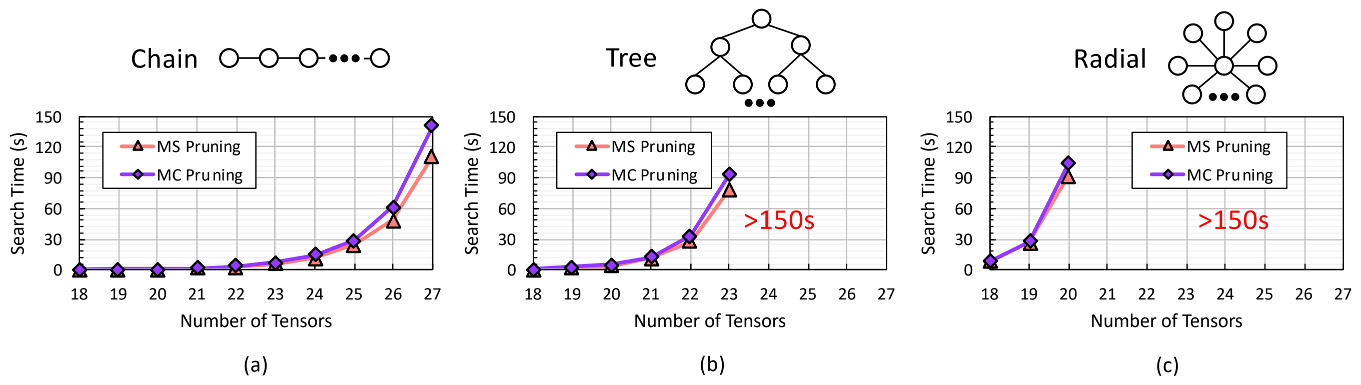


Figure 9: Search time comparison on different network topologies without extra edges after applying the outer product pruning: (a) Chain topology; (b) Tree topology; (c) Radial topology. In order to keep the same time range across all cases, we do not show the search time larger than 150 seconds.

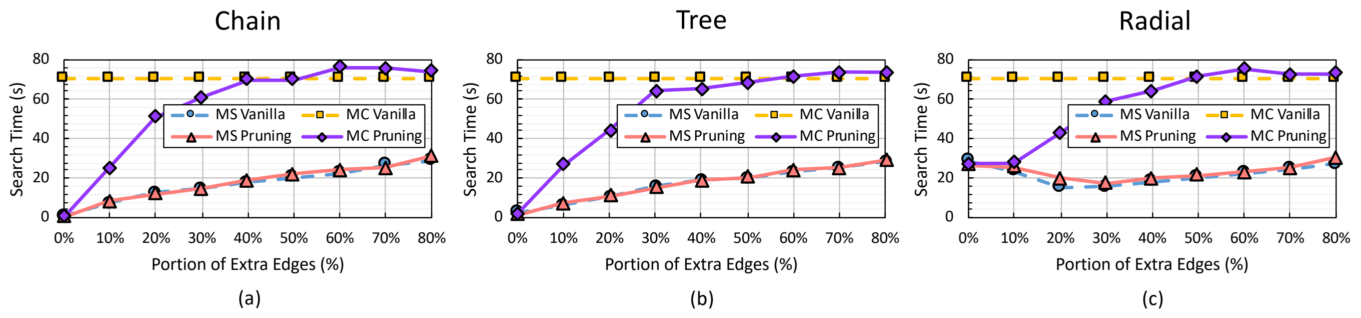


Figure 10: Search time comparison on different network topologies with different portions of extra edges: (a) Chain topology; (b) Tree topology; (c) Radial Topology. Here each network contains 19 tensors.

in the vanilla search. Second, with our reduction optimization, the search on the chain topology goes the fastest while on the radial topology is the slowest. On the chain topology, it has the most prunable possible tensors, leading to a high reduction ratio of the search space; in contrast, the number of prunable possible tensors greatly decreases on the radial topology, which still has a large search space after applying the outer product pruning.

The results after applying our pruning strategy on the networks with different portions of extra edges are shown in Figure 10. We first analyze the performance when the measurement metric is MC. We find that the search time is similar among the tensor networks with different basic topologies after we introduce extra edges. The reason is that when we add extra edges, the basic topology impacts less on the number of prunable possible tensors. Moreover, the vanilla algorithm goes through all possible tensors and all split cases no matter if the possible tensors can be pruned from the search space. Thus, the search time keeps unchanged even if with different portions of extra edges in the network. The last observation is that when the portion of extra edges is smaller than 50%, the search can take benefit from applying our search space reduction; however, as the portion of extra edges becomes larger, the search time with space reduction even exceeds the vanilla algorithm without space reduction. This is because the reduction overhead fails to cover its benefit when the network is sufficiently dense with a very limited reduction ratio.

Also in Figure 10, we analyze the performance when the measurement metric is MS. Different from the results of the observation in searching the lowest MC, the search time of the vanilla search varies as extra edges increase. This is because the vanilla search of the lowest MS does not go through all split cases due to the early stop mechanism, and the increased edges can impact the early stop time. Second, there is no benefit from applying the outer product pruning after we introduce extra edges into the network. This is because the calculation under the MS metric is cheaper, which makes the overhead introduced by the outer product pruning relatively more heavy than that under the MC metric. At last, when the basic topology is radial, the search time is reduced when we introduce <20% extra edges. This phenomena is again caused by the early stop mechanism. For the radial topology with a small number of extra edges, the center tensor in the original network often consumes the largest SE. Thus, there is no early stop for the possible tensors which involve the center tensor during search (i.e. the conditions of line 32-34 in Algorithm 2 or line 41-43 in Algorithm 4 will not happen). This situation can be alleviated by introducing more extra edges.

Except the above network topologies, we also evaluate our search methods on grid topology which is an important case in both physics and machine learning communities. We estimate the performance on three different grid sizes in Figure 11(a). Based on the search time result, the search complexity of the grid topology lies between the tree topology and radius topology which is determined by the number of prunable possible tensors. Then we introduce different portions of extra edges in the network under the grid size of 3×8. The result is depicted in Figure 11(b), which presents similar characters as on aforementioned topologies.

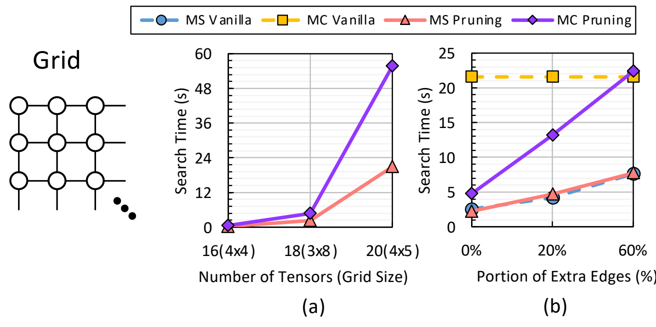


Figure 11: Search time for the network with grid topology after applying outer product pruning: (a) Search time without extra edges under different grid sizes; (b) Search time with different portions of extra edges under grid size of 3×8.

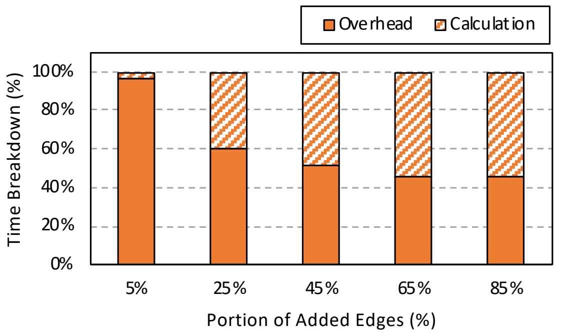


Figure 12: Search time breakdown on a tree network with 19 tensors under different portions of extra edges. Here the measurement metric is MC and we include the outer product pruning during search.

Figure 12 presents the search time breakdown on a tree network with 19 tensors under different portions of extra edges. The calculation part includes the sharing order calculation and the contraction expense calculation, and the overhead includes other operations like traversing all possible tensors and split cases, finding prunable possible tensors, data loads, etc. When the measurement metric is MS, the sharing order calculation and the contraction expense calculation only happen once for each possible tensor. So we only evaluate the time breakdown when the measurement metric is MC. The total overhead time can be acquired by removing line 32-35 in Algorithm 4. From the results, the portion of calculation time increases from 3.9% to 53.8% as more extra edges are added into the tensor network. More extra edges can reduce the number of prunable possible tensors and increase the search space. Hence, more search time will be spent on calculation. Also, we find that the ratio of calculation time is close when we add 65% and 85% extra edges. The reason is that the number of prunable possible tensors in later $Sets$ are similar in these two cases, and the possible tensors in later $Sets$ occupy majority of the search time since they have a larger number of split cases compared with the possible tensors in earlier $Sets$.

D. Parallelism Analysis

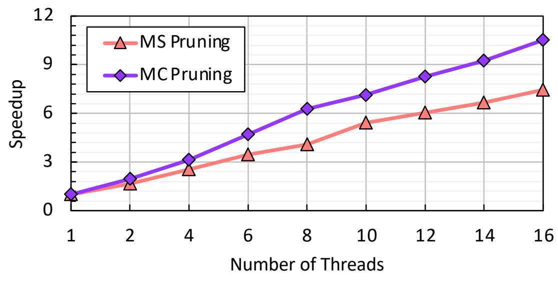


Figure 13: Speedup over the single-thread implementation through parallelism optimization on a binary tree with 19 tensors and 25% extra edges.

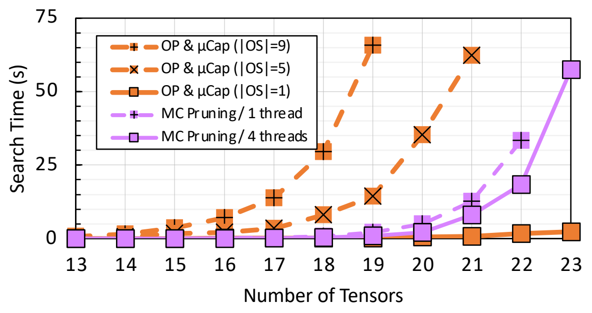
We evaluate the speedup after using the multithread optimization in Figure 13. The testing network is a binary tree with 19 tensors and 25% extra edges, and the outer product pruning is used during search. As aforementioned, different from other subsections, here we adopt an Intel Xeon processor to support more threads. We find that multithread optimization gains a higher speedup when the measurement metric is MC since the portion of overhead in MC is smaller. Furthermore, as the number of threads increases, the speedup grows but eventually saturated. This is because other overhead caused by the multithread implementation begins to matter when the number of threads reaches a threshold.

E. Comparison with Prior Work

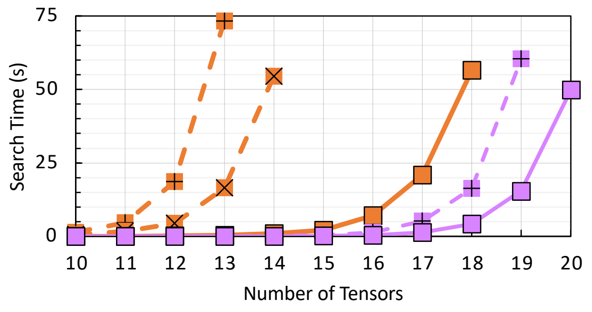
Some prior work try to find an approximate optimal contraction sequence of a tensor network [29], [32] or target on a specific network topology [31]. Our algorithms can find the exactly optimal solution for arbitrary network. For the prior work that consider the compute cost of the contraction sequence also have some constraints on the tensor network. For example, the search on networks without free orders is targeted in [30]; although a polynomial solution is provided in [31], the networks are restricted in the tree topology.

One of the closest prior work is OP & μ_{Cap} [27], which considers the compute cost of a given contraction sequence. Instead of finding a sequence with the least MC, the measurement metric in [27] is the total CE of a contraction sequence. Note that maximum CE is usually close to total CE as the maximum CE is often much larger than the CE in other steps. Since the measurement metrics of this work and the prior work in [27] are actually different, we just present a coarse comparison for interesting insights in this subsection rather than intending to beat it.

We compare our pruning method with OP & μ_{Cap} on a binary tree without extra edges. The comparison is shown in Figure 14. Here we define the order set as $OS = \{O_1, O_2, ..., O_n\}$. Each free or share order is randomly chosen from OS. We use $|OS|$ to denote the size of order set, and we have $|OS| = n$ here. We find that the size of OS affects



(a)



(b)

Figure 14: Search time comparison between our method and the OP & μ_{Cap} [27] on a binary tree (a) without and (b) with 25% extra edges.

the search time of OP & μ_{Cap} a lot. Although when $|OS|$ is very small, e.g., $|OS| = 1$, $OP\&\mu_{Cap}$ is faster than our method, the performance degrades dramatically and becomes much worse than ours as $|OS|$ increases. Compared with [27], our method will not be affected by the value of $|OS|$. This is because, when we consider MS, the search time is affected by the number of prunable possible tensors rather than $|OS|$. When we include 25% extra edges, all the performance results degrade due to the larger search space. Notice that in this case, our method behaves faster at all $|OS|$ settings. This indicates that when the tensor network structure does not follow a regular pattern, our search algorithm can be a better choice.

V. Conclusion

In this work we focus on the acceleration of searching an optimal contraction sequence with the lowest MS or MC. A data format based on log_k representation and data structure based on adjacency matrix with additional intermediate vectors are designed for efficient computation. We further incorporate the outer product pruning into the BFS search to reduce the search space. At last, in the execution level, we implement the multithread optimization to improve the parallelism.

From the performance analyses under different basic topology, network scale, and portion of extra edges, we provide several insights as follows. (1) The evaluation metric impacts the search algorithm and the search time. For example, the search time would become smaller when the evaluation metric of the contraction sequence is the maximum storage expense, due to the less computation and the early stop mechanism. (2) The data structure and hardware-level support indeed help. The adjacency matrix based design gains acceleration and the parallel execution further improves the performance. Moreover, the data access in our design is discontinuous, which may decrease the cache hit rate when the network scales up. The hardware-level architectural design might help in the future. (3) The tensor network topology also matters. In sparse networks, when the measurement metric is MC, the outer product pruning method can gain more benefits, and the chain/tree topology with fewer connections gives more benefits than the radial topology. However, in dense networks with increased connections, we recommend not to use any space reduction technique that usually pays unaffordable extra overhead, and the performance gap between basic topologies will be narrowed. In reality, it is possible to design specific algorithms according to the practical topologies.

References

[1] Y.-Y. Shi, L.-M. Duan, and G. Vidal, “Classical simulation of quantum many-body systems with a tree tensor network,” *Physical Review A*, vol. 74, no. 2, p. 022320, 2006.

[2] A. Feiguin, S. Trebst, A. W. Ludwig, M. Troyer, A. Kitaev, Z. Wang, and M. H. Freedman, “Interacting anyons in topological quantum liquids: The golden chain,” *Physical Review Letters*, vol. 98, no. 16, p. 160409, 2007.

[3] G. Vidal, “Class of quantum many-body states that can be efficiently simulated,” *Physical Review Letters*, vol. 101, no. 11, p. 110501, 2008.

[4] P. Corboz, M. Lajkó, A. M. Läuchli, K. Penc, and F. Mila, “Spin-orbital quantum liquid on the honeycomb lattice,” *Physical Review X*, vol. 2, no. 4, p. 041013, 2012.

[5] J. Eisert, M. Friesdorf, and C. Gogolin, “Quantum many-body systems out of equilibrium,” *Nature Physics*, vol. 11, no. 2, p. 124, 2015.

[6] F. Verstraete, V. Murg, and J. I. Cirac, “Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems,” *Advances in Physics*, vol. 57, no. 2, pp. 143–224, 2008.

[7] G. Evenbly and G. Vidal, “Entanglement renormalization in two spatial dimensions,” *Physical Review Letters*, vol. 102, no. 18, p. 180406, 2009.

[8] R. N. Pfeifer, G. Evenbly, and G. Vidal, “Entanglement renormalization, scale invariance, and quantum criticality,” *Physical Review A*, vol. 79, no. 4, p. 040301, 2009.

[9] P. Corboz, S. R. White, G. Vidal, and M. Troyer, “Stripes in the two-dimensional t-j model with infinite projected entangled-pair states,” *Physical Review B*, vol. 84, no. 4, p. 041108, 2011.

[10] R. Orús, “A practical introduction to tensor networks: Matrix product states and projected entangled pair states,” *Annals of Physics*, vol. 349, pp. 117–158, 2014.

[11] S. Szalay, M. Pfeffer, V. Murg, G. Barcza, F. Verstraete, R. Schneider, and Ö. Legeza, “Tensor product methods and entanglement optimization for ab initio quantum chemistry,” *International Journal of Quantum Chemistry*, vol. 115, no. 19, pp. 1342–1391, 2015.

[12] L. Cincio, J. Dziarmaga, and M. M. Rams, “Multiscale entanglement renormalization ansatz in two dimensions: quantum ising model,” *Physical Review Letters*, vol. 100, no. 24, p. 240603, 2008.

[13] P. Corboz and G. Vidal, “Fermionic multiscale entanglement renormalization ansatz,” *Physical Review B*, vol. 80, no. 16, p. 165129, 2009.

[14] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, “Simulation of low-depth quantum circuits as complex undirected graphical models,” *arXiv preprint arXiv:1712.05384*, 2017.

[15] J. Chen, F. Zhang, M. Chen, C. Huang, M. Newman, and Y. Shi, “Classical simulation of intermediate-size quantum circuits,” *arXiv preprint arXiv:1805.01450*, 2018.

[16] Z. Zhang, T.-W. Weng, and L. Daniel, “Big-data tensor recovery for high-dimensional uncertainty quantification of process variations,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 7, no. 5, pp. 687–697, 2017.

[17] Z. Zhang, K. Batselier, H. Liu, L. Daniel, and N. Wong, “Tensor computation: A new framework for high-dimensional problems in eda,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 4, pp. 521–536, 2017.

[18] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, “Tensorizing neural networks,” in *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

[19] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: compressing convolutional and fc layers alike," *arXiv preprint arXiv:1611.03214*, 2016.

[20] Y. Yang, D. Krompass, and V. Tresp, "Tensor-train recurrent neural networks for video classification," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3891–3900, JMLR. org, 2017.

[21] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[22] C. I. Kanatsoulis, X. Fu, N. D. Sidiropoulos, and M. Akçakaya, "Tensor completion from regular sub-nyquist samples," *IEEE Transactions on Signal Processing*, vol. 68, pp. 1–16, 2019.

[23] W. Hackbusch, *Tensor spaces and numerical tensor calculus*, vol. 42. Springer Science & Business Media, 2012.

[24] L. Chi-Chung, P. Sadayappan, and R. Wenger, "On optimizing a class of multi-dimensional loops with reduction for parallel execution," *Parallel Processing Letters*, vol. 7, no. 02, pp. 157–168, 1997.

[25] I. L. Markov and Y. Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, 2008.

[26] S. Hirata, "Tensor contraction engine: Abstraction and automated parallel implementation of configuration-interaction, coupled-cluster, and many-body perturbation theories," *The Journal of Physical Chemistry A*, vol. 107, no. 46, pp. 9887–9897, 2003.

[27] R. N. Pfeifer, J. Haegeman, and F. Verstraete, "Faster identification of optimal contraction sequences for tensor networks," *Physical Review E*, vol. 90, no. 3, p. 033315, 2014.

[28] S. Kourtis, C. Chamon, E. R. Mucciolo, and A. E. Ruckenstein, "Fast counting with tensor networks," *arXiv preprint arXiv:1805.00475*, 2018.

[29] A. S. Jermyn, "Automatic contraction of unstructured tensor networks," *arXiv preprint arXiv:1709.03080*, 2017.

[30] G. Evenbly and R. N. Pfeifer, "Improving the efficiency of variational tensor network algorithms," *Physical Review B*, vol. 89, no. 24, p. 245118, 2014.

[31] J. Xu, L. Liang, L. Deng, C. Wen, Y. Xie, and G. Li, "Towards a polynomial algorithm for optimal contraction sequence of tensor networks from trees," *Physical Review E*, vol. 100, no. 4, p. 043309, 2019.

[32] A. Hartono, Q. Lu, X. Gao, S. Krishnamoorthy, M. Nooijen, G. Baumgartner, D. E. Bernholdt, V. Choppella, R. M. Pitzer, J. Ramanujam, *et al.*, "Identifying cost-effective common subexpressions to reduce operation count in tensor contraction evaluations," in *International Conference on Computational Science*, pp. 267–275, Springer, 2006.

[33] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.