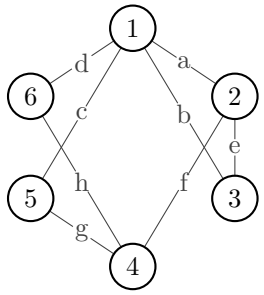
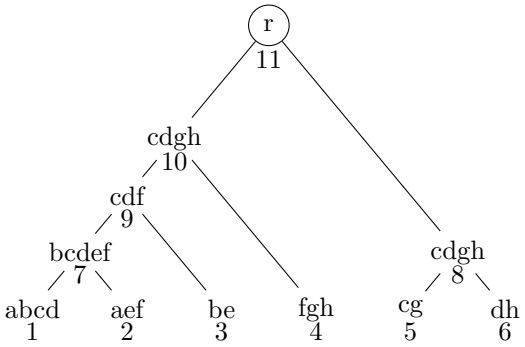
Christian B. Mendl, Richard M. Milbradt

Tutorial 12 (Contraction trees[1])

Let us abstractly represent a tensor network as graph (V, E), where the vertices $V = (v_1, v_2, \ldots, v_m)$ stand for the tensors and the edges $E = \{(v_i, v_j) : v_i, v_j \in V \text{ connected}, i < j\}$ the to-be contracted legs. For simplicity we assume that there are no open legs. The function $\dim : E \to \mathbb{N}$ specifies the dimension of a given edge. As shown in the example on the right, we label vertices by numbers and edges by roman letters.



A *contraction tree* is an undirected edge-weighted binary tree and describes the process of contracting a tensor network. Formally, $T = (U, E_{\text{tree}})$, where $U = (u_1, u_2, \ldots, u_p)$ is a set of nodes and E_{tree} the set of edges.



The tree is constructed as follows:

1. Each vertex in the tensor network is a leaf node in the contraction tree. For convenience, one identifies v_1 with u_1, v_2 with u_2 and so on. We introduce an injective function $\ell : U \to S$, where S contains sets of all possible combinations of edges in E. For the leaf nodes, ℓ returns the legs (connected edges) of the corresponding tensor, e.g., $\ell(u_1) = \{a, b, c, d\}$.

2. We then choose any two orphaned nodes and insert a parent node above until only a single *root node* remains. ℓ of the parent is the symmetric difference (disjunctive union) of ℓ of the child nodes, i.e., for the parent of nodes u_i and u_j:

$$\ell(u_{\text{parent}}) = (\ell(u_i) \cup \ell(u_j)) \backslash (\ell(u_i) \cap \ell(u_j)).$$

   Intuitively, a parent node results from the contraction of the two child nodes, and ℓ enumerates the legs of each (intermediate) tensor.

3. We calculate the weight of each edge in the tree. The weight is the product of the dimensions of the node on the edge farther away from the root:

$$\text{con}((u_i, u_{\text{parent}})) = \prod_{e \in \ell(u_i)} \dim(e).$$

   We refer to this quantity as the *congestion* of the edge (equivalent to the dimension connecting u_i to the rest of the network, i.e., the overall number of entries of u_i).

   The *congestion* of a node is similar but considers legs in child nodes as well:

$$\text{con}(u_i) = \prod_{e \in D} \dim(e), \quad \text{where } D = \bigcup_{u \in \text{child}(u_i)} \ell(u)$$

For simplicity, we assume uniform bond dimensions in the following, i.e., $\dim(e) = n$ for all $e \in E$.

(a) Each non-leaf node in the tree represents a contraction operation. Based on the defined quantities, how can one determine the spatial (memory) and temporal (arithmetic) cost of a contraction?

(b) Estimate the spatial and temporal cost associated with the above contraction tree.

(c) Find a contraction tree that is spatially and temporally more efficient.[2]

(d) Construct another contraction tree if we only consider time-to-solution, and can use parallel computation.

[1] cf. B. O'Gorman: *Parameterization of tensor network contraction*, TQC 2019 vol. 135, 10:1–10:19 (arXiv:1906.00013) and J. Gray, S. Kourtis: *Hyper-optimized tensor network contraction*, Quantum 5, 410 (2021) (arXiv:2002.01935)

[2] Finding the optimal contraction sequence for a general tensor network is a NP-hard problem, see R. Pfeifer, J. Haegeman, F. Verstraete: *Faster identification of optimal contraction sequences for tensor networks*, PRE 90, 033315 (2014) (arXiv:1304.6112)

Solution

(a) With some observation, we can determine that the memory requirement of each contraction operation is equivalent to the sum of congestions of each edge connected to the corresponding node in the contraction tree. For example, the memory requirement for the contraction at node 7 is

$$\begin{aligned}\text{mem} &= \mathcal{O}(\text{con}((u_1, u_7)) + \text{con}((u_2, u_7)) + \text{con}((u_7, u_9))) \\ &= \mathcal{O}(\dim(a)\dim(b)\dim(c)\dim(d) + \dim(a)\dim(e)\dim(f) + \dim(b)\dim(c)\dim(d)\dim(e)\dim(f)) \\ &= \mathcal{O}(n^4 + n^3 + n^5)\end{aligned}$$

The result is $\mathcal{O}(n^5)$, combining the dimensions (and thus the memory requirement) of the tensors before and after contraction.

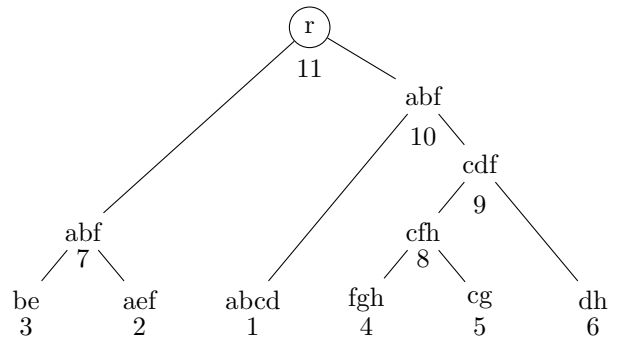
Additionally, the complexity of the operation is then the congestion of the node itself.

$$\begin{aligned}\text{complexity} &= \mathcal{O}(\text{con}(u_7)) \\ &= \mathcal{O}(\dim(a)\dim(b)\dim(c)\dim(d)\dim(e)\dim(f)) \\ &= \mathcal{O}(n^6)\end{aligned}$$

The complexity of the operation is of order $\mathcal{O}(n^6)$.

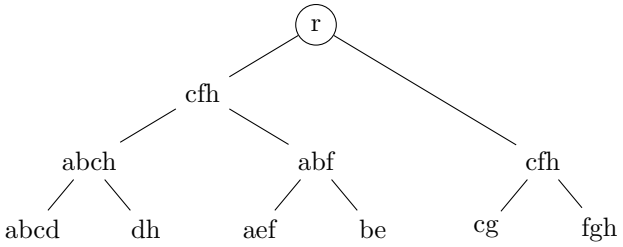
(b) We can immediately see from the labels that the first contraction operation has both the highest complexity and memory requirement. The memory requirement is $\mathcal{O}(n^5)$ and the complexity is $\mathcal{O}(n^6)$.

(c) In order to ensure memory efficiency, we try to create labels as short as possible at each contraction. In order to reduce the complexity of each operation, we try to ensure that the number of dimensions involved in each contraction is minimized. In order to do so, let us use a greedy algorithm that always selects the operation with the lowest memory cost and chooses randomly if there is a tie.



We see that this immediately lowers the memory requirement to $\mathcal{O}(n^4)$ and the time complexity to $\mathcal{O}(n^5)$.

(d) Other than lower the complexity of each operation, we can also introduce parallelism to the contraction operations. We use an algorithm that forces a parallel contraction whenever possible, minimizing memory costs whenever possible.



We see that while the largest contraction operations remains $\mathcal{O}(n^5)$, we can now perform multiple operations in parallel, speeding up the time to solution. However, this comes at the cost of storing another tensor with $\mathcal{O}(n^4)$ entries.