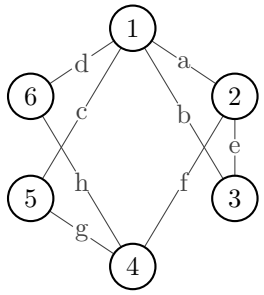
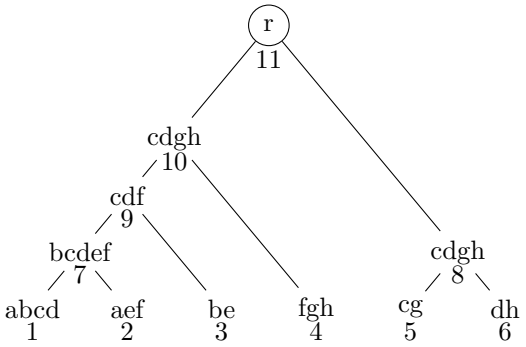
Christian B. Mendl, Richard M. Milbradt due: 21 July 2022 (before tutorial)

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.

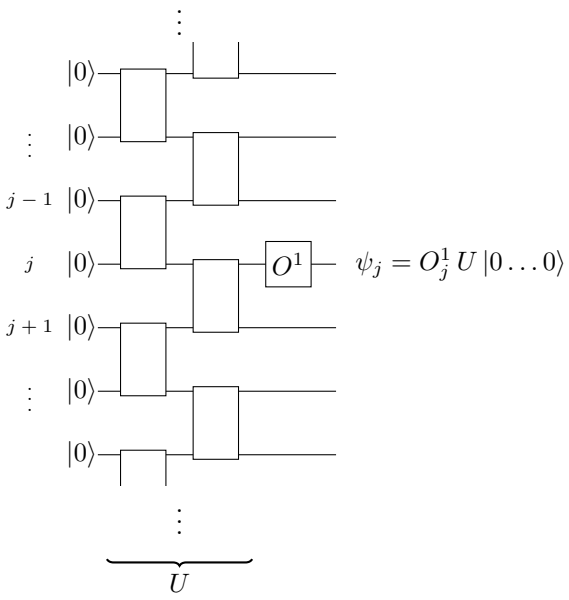
[1] cf. B. O'Gorman: *Parameterization of tensor network contraction*, TQC 2019 vol. 135, 10:1–10:19 (arXiv:1906.00013) and J. Gray, S. Kourtis: *Hyper-optimized tensor network contraction*, Quantum 5, 410 (2021) (arXiv:2002.01935)

[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)

Exercise 12.1 (Light cone pattern of dynamical correlations in brick wall quantum circuits)

A common quantum circuit layout is a "brick wall" arrangement of two-qubit gates, as illustrated on the right. One can interpret such a layout as discretization of a quantum time evolution step, $U \approx \mathrm{e}^{-iH\Delta t}$ (cf. the TEBD algorithm).

Our goal here is to compute the so-called *dynamical correlation function* between the j-th and k-th qubit ($k \geq j$). This is achieved via the following construction: The initial state $|0\ldots0\rangle$ time-evolves for a step Δt; we then apply a single-qubit gate O^1 to the j-qubit. O^1 is also assumed to be Hermitian, like for example one of the Pauli matrices, and plays the role of an observable. We denote the overall output state of this protocol by ψ_j, as illustrated on the right. (The total number of qubits is assumed to be large enough such that boundary effects are not relevant. The following considerations are valid even if all the two-qubit gates are different.)



Regarding the k-th qubit, we again start from the state $|0\ldots0\rangle$, then first apply another gate O^2 to the k-qubit, and finally perform a time step. The overall output of this protocol is thus $\chi_k = UO_k^2\,|0\ldots0\rangle$.

The dynamical correlation function[3] is now defined as

$$c_{j,k} = \langle \psi_j, \chi_k \rangle .$$

(a) Draw the diagram for evaluating $c_{j,k}$ for the cases $k = j+2$ and $k = j+3$, and simplify it as much as possible by canceling two-qubit gates (i.e., a two-qubit gate V directly followed by $V^\dagger$).

(b) For $k = j+3$, compute the so-called connected correlation function

$$c_{j,k}^{\mathrm{conn}} = c_{j,k} - \langle 0\ldots0|\, U^\dagger O_j^1 U\, |0\ldots0\rangle \cdot \langle 0\ldots0|\, O_k^2\, |0\ldots0\rangle\,.$$

Does your result change when considering $k \geq j+3$ in general?

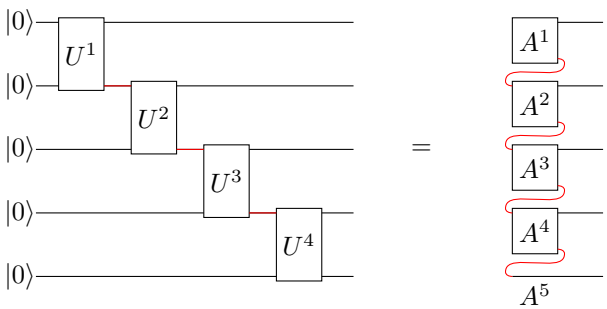
(c) (Voluntary) Repeat the calculations using two time steps, by applying U twice at each occurrence.

Remark: You should learn form this exercise that the brick wall pattern leads to a "light cone" spreading of correlations, i.e., qubits j and k can only be correlated (non-zero $c_{j,k}^{\mathrm{conn}}$) if there are sufficiently many time steps to reach k from j.

Exercise 12.2 (Shallow quantum circuit simulation)

The goal of this exercise is to convert the output quantum state of certain shallow-type[4] quantum circuits into MPS form (which can then be used to compute expectation values of observables, for example).

A quantum circuit applying a shifted sequence of two-qubit gates is already of MPS form when absorbing the inputs $|0\rangle$ into the gates (interpreted as tensors) and reinterpreting the connecting wires as virtual bonds. As example (virtual bonds highlighted in red):



Note that the last MPS tensor simply acts as identity.

[3] In the physics literature, the correlation function is typically written as $c_{j,k} = \left\langle O_j^1(\Delta t)\, O_k^2 \right\rangle$, with $O_j^1(\Delta t) = U^\dagger O_j^1 U$ the time-evolved operator in the Heisenberg picture.

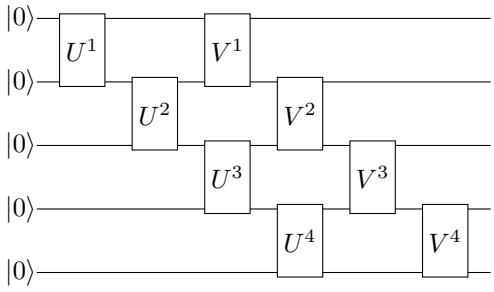
[4] Strictly speaking, these are not shallow circuits according to the definition in the lecture. What we really exploit is the one-dimensional arrangement of qubits, and that only a few gates act on each neighboring qubit pair.

(a) Write a Python function `shifted_gate_sequence_to_mps(Ulist)` which takes a list of unitary gates $(U^1, U^2, \ldots, U^{d-1})$ as input, and returns a corresponding MPS (Python class from the Jupyter notebook template for this exercise). The class definition now additionally contains a "classmethod" for initializing an object via a list of MPS tensors.

Hint: First reshape each U^j gate into a $2 \times 2 \times 2 \times 2$ tensor. Then label the dimensions of U^j (in the correct order!) in the circuit diagram. The application to $|0\rangle$ corresponds to selecting the subtensor $U^j_{:,:,:,0}$.

(b) Calculate (with "pen and paper") the output quantum state when setting $U^1 = U_{\mathrm{CNOT}} \cdot (H \otimes I)$, where H is the Hadamard gate, and $U^j = U_{\mathrm{CNOT}}$ for all $j \geq 2$. Then use this state to test your implementation from (a): calculate the MPS representation for these gates, convert it to a statevector via `as_vector()`, and ensure this vector agrees with your analytical result.

(c) (Voluntary) Write a function to convert a circuit with the following topology to a MPS:



Hint: You can use your implementation from (a) for the sequence of U^j gates, and then apply the V^j gates to this intermediate MPS representation, using similar techniques as for the TEBD algorithm.