Technische Universität München
Fakultät für Physik

Abschlussarbeit im Masterstudiengang Physik der Kondensierten Materie

Entwicklung eines diagonalen isometrischen Tensor Netzwerk Algorithmus

Development of a diagonal isometric Tensor Network Algorithm

Benjamin Sappler

18. April 2024

Erstgutachter (Themensteller): Prof. F. Pollmann
Zweitgutachter: Unknown

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 99.99.2099 Benjamin Sappler

Abstract

The numerical simulation of strongly interacting quantum many-body systems is a challenging problem. In the last decades, Tensor Networks have emerged as the standard method for tackling this problem in one dimensional systems in the form of Matrix Product States (MPS). Tensor Networks have also been generalized for the highly relevant problem of two and more spatial dimensions. However, these so-called Projected Entangled Pair States (PEPS) are typically plagued by high computational complexity or drastic approximations. Recently, a new class of Tensor Networks, called isometric Tensor Networks, have been proposed for the simulation of two-dimensional quantum systems. This new class of Tensor Networks can be understood as a generalization of the one-dimensional Matrix Product States to higher dimensions. While isometric Tensor Networks generally capture only a subspace of the total Hilbert space, there are already promising results. In this work, we develop a new class of isometric Tensor Networks that has some key differences to the existing one. We show first numerical results for finding ground states of the Transverse Field Ising model.

Zusammenfassung

Contents

Chapter 1

Introduction

Chapter 2

Tensors and Tensor Networks

In the following, a brief introduction to tensors, tensor networks, and tensor network algorithms is given. We start by defining the conventions and notation used in this thesis in section 2.1. In section 2.2, we introduce important tensor decompositions that are used extensively in tensor network algorithms. In section 2.3, we define isometric tensor networks and discuss their properties. Lastly, we give examples for physical states being represented in terms of isometric tensor networks, namely the popular Matrix Product States (MPS) in section 2.4 and the recently developed isometric tensor product states in 2D (isoTPS) in section 2.5.

2.1 Conventions and Notation

Tensors

For the purpose of this thesis we define a *tensor T of rank n* as an n-dimensional array of complex numbers

$$T \in \mathbb{C}^{\chi_1 \times \chi_2 \times \cdots \times \chi_n}, \quad \chi_i \in \{1, 2, \dots\}$$

with entries

$$T_{i_1 i_2 \dots i_n} \in \mathbb{C}, \quad i_j \in \{1, 2, \dots, \chi_j\}.$$

For example, a rank-0 tensor is a scalar, a rank-1 tensor is a vector, and a tensor of rank-2 is a matrix.

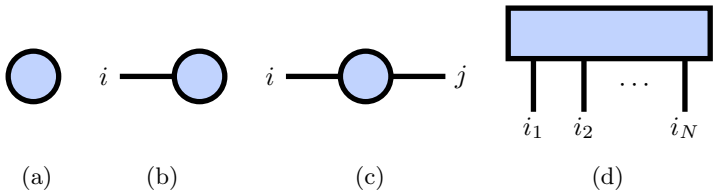


Figure 2.1: Tensors of different ranks are shown in diagrammatic notation. (a) A scalar, (b) a vector, (c) a matrix, (d) a general tensor of rank N.

An *index contraction* between two or more tensors is the linear operation that is performed by summing over a given set of indices. For example, the matrix product of two matrices $A \in \mathbb{C}^{\chi_1 \times \chi_2}$ and $B \in \mathbb{C}^{\chi_2 \times \chi_3}$ can be written as the index contraction

$$C_{ij} = \sum_{\alpha=1}^{\chi_2} A_{i\alpha} B_{\alpha j}. \tag{2.1}$$

A more involved example is the index contraction of two rank-3 tensors $A \in \mathbb{C}^{\chi_1 \times \chi_2 \times \chi_3}$, $B \in \mathbb{C}^{\chi_2 \times \chi_4 \times \chi_5}$ and one rank-4 tensor $C \in \mathbb{C}^{\chi_3 \times \chi_5 \times \chi_6 \times \chi_7}$, where we contract along the indices with dimension χ_2, χ_3 and χ_5. The result is a rank-4 tensor $D \in \mathbb{C}^{\chi_1 \times \chi_4 \times \chi_6 \times \chi_7}$:

$$D_{ijkl} = \sum_{\alpha=1}^{\chi_2} \sum_{\beta=1}^{\chi_3} \sum_{\gamma=1}^{\chi_5} A_{i\alpha\beta} B_{\alpha j\gamma} C_{\beta\gamma kl}. \tag{2.2}$$

More generally, given a rank-$(n+f)$ tensor $A \in \mathbb{C}^{\chi_1 \times \cdots \times \chi_n \times \xi_1 \times \cdots \times \xi_f}$ and a rank-$(m+f)$ tensor $B \in \mathbb{C}^{\lambda_1 \times \cdots \times \lambda_m \times \xi_1 \times \cdots \times \xi_f}$, the result of contracting A and B along the last f indices produces a new rank-$(m+n)$ tensor $C \in \mathbb{C}^{\chi_1 \times \cdots \times \chi_n \times \lambda_1 \times \cdots \times \lambda_m}$ as

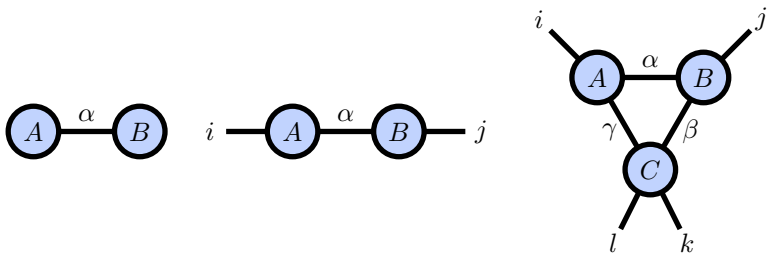
$$C_{i_1 \ldots i_n j_1 \ldots j_m} \coloneqq \sum_{\mu_1=1}^{\xi_1} \cdots \sum_{\mu_f}^{\xi_f} A_{i_1 \ldots i_n \mu_1 \ldots \mu_f} B_{j_1 \ldots j_m \mu_1 \ldots \mu_f}. \tag{2.3}$$

Contractions over arbitrary indices can be reformulated as contractions over the last f indices by transposing the tensors. Contractions over more than two tensors can be decomposed into successive contractions of two tensors. Because tensor contractions are linear, the order in which tensors are contracted doesn't change the result. However, the computational complexity does in general depend on the order of contractions and can thus be minimized by choosing the optimal contraction order.
By counting the number of multiplications and additions that are necessary to perform the tensor contraction (2.3), the computational complexity can be determined as

$$\mathcal{O}\left(\prod_{\mu=1}^{n} \chi_\mu \prod_{\mu=1}^{m} \lambda_\mu \prod_{\mu=1}^{f} \xi_f \right).$$

Tensor Networks

A *tensor network* is defined as a collection of tensors that are contracted in a specified way. It is convenient to introduce a diagrammatic notation, where tensors are drawn as shapes and tensor indices are represented by lines emerging from these



(a) (b) (c)

Figure 2.2: Tensor networks in diagrammatic notation. (a) Scalar product (2.4). (b) Matrix product (2.1). (c) More involved network consisting of three tensors (2.2).

shapes. To relate this diagrammatic notation to equations, one often decorates each line with the corresponding index i_j. A scalar, vector, matrix, and a general rank-n tensor are visualized in this notation in figure 2.1. Index contractions are depicted diagrammatically by connecting the lines corresponding to contracted indices. Lines connecting two tensors are sometimes called *bonds*, while indices not used in contractions are called *open indices*. The *bond dimension* χ_i denotes the number of different values an index i can take. It is often more convenient to discuss tensor network algorithms in terms of diagrams than in terms of equations. Two simple tensor networks are the scalar-product of two vectors

$$c = \sum_{\alpha=1}^{\chi} A_\alpha B_\alpha \tag{2.4}$$

and the matrix product (2.1). Both networks are shown as diagrams in figure 2.2(a) and 2.2(b) respectively. 2.2(c) depicts the more involved tensor network (2.2).

Isometries

Given two normed vector spaces V_1 and V_2 with $\dim(V_1) = m$, $\dim(V_2) = n$, $m \leq n$, an *isometry* (sometimes also called *semi-unitary*) is a linear, norm-preserving map $W : V_1 \to V_2$ from the smaller to the larger vector space. Each isometry can be represented by a $n \times m$ matrix W fulfilling the *isometry condition*

$$W^\dagger W = \mathbb{1}, \quad WW^\dagger = \mathbb{P}, \tag{2.5}$$

where $\mathbb{P} = \mathbb{P}^2$ is a projection. If $m = n$, it holds $\mathbb{P} = \mathbb{1}$ and W is a *unitary map*. An isometry tensor is a tensor that through grouping of indices and reshaping (i.e. matricization) becomes an isometry. In tensor network diagrams, we draw isometries by decorating lines with arrows. Following the convention of [1, 2], we denote the

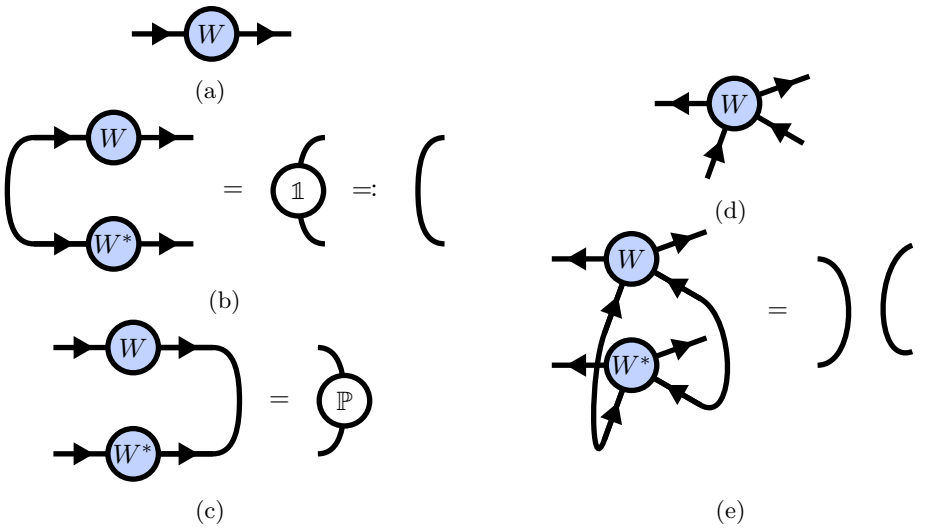


Figure 2.3: Isometric and unitary tensors are drawn by decorating indices with arrows. (a) diagrammatic notation of an isometric matrix (b) (c) The isometry condition (2.5) is depicted diagrammatically. (d) Isometric tensors of higher rank must fulfill the isometry condition by grouping of indices.

indices belonging to the larger vector space by incoming arrows and the indices belonging to the smaller vector space by outgoing arrows. Unitary tensors are decorated with bidirectional arrows on all indices, where the grouping must be inferred from the context. Ordinary tensors are drawn without arrows. Tensor diagrams for isometric and unitary tensors are shown in figure 2.3.
We lastly introduce an inner product for rank-n tensors $A, B \in \mathbb{C}^{\chi_1 \times \cdots \times \chi_n}$, the *Frobenius inner product*

$$\langle A, B \rangle_{\mathrm{F}} \coloneqq \sum_{\mu_1=1}^{\chi_1} \cdots \sum_{\mu_n=1}^{\chi_n} A^*_{\mu_1 \ldots \mu_n} B_{\mu_1 \ldots \mu_n} = \mathrm{Tr}\left(A^\dagger B\right),$$

where the last equality holds only if $n = 2$. The Frobenius inner product induces a norm, the *Frobenius norm*

$$\|A\|_{\mathrm{F}} = \sqrt{\langle A, A \rangle_{\mathrm{F}}},$$

which can be used to define a measure of distance $\|A - B\|_{\mathrm{F}}$ between tensors A and B.

2.2 Tensor Decompositions

There are two decompositions that are used extensively in this thesis: The QR-decomposition and the Singular Value Decomposition. Both decompositions are matrix decompositions but can be applied to tensors as well by first grouping indices and reshaping to a matrix, applying the decomposition, and reshaping the result back to the original bond dimensions.
The *reduced QR-decomposition* of a matrix $A \in \mathbb{C}^{n\times m}$ is the decomposition

$$A = QR, \tag{2.6}$$

where $Q \in \mathbb{C}^{n\times k}$ is an isometry, $R \in \mathbb{C}^{k\times m}$ is an upper triangular matrix and $k \coloneqq \min(n, m)$. The computational complexity of the QR decomposition scales as

$$\mathcal{O}\left(n \cdot m \cdot \min(n, m)\right). \tag{2.7}$$

A diagrammatic depiction of the QR decomposition (2.6) is drawn in figure 2.4(a).
The *Singular Value Decomposition* (SVD) of a matrix $A \in \mathbb{C}^{n\times m}$ is the decomposition

$$A = USV^\dagger, \tag{2.8}$$

where $U \in \mathbb{C}^{n\times k}$ and $V \in \mathbb{C}^{m\times k}$ are isometries, $S \in \mathbb{R}^{k\times k}$ is a diagonal real matrix of *singular values*, and $k \coloneqq \min(n, m)$. The computational complexity of the SVD is the same as for the QR decomposition (2.7). However, while the scaling is the same, the prefactors are lower for the QR decomposition in most implementations, meaning that the QR decomposition is faster in practice. Moreover, in contrast to the SVD, the QR decomposition allows for highly efficient implementations on graphics processing units (GPUs), which enables decompositions of large matrices to be carried out significantly faster and more power efficiently. Thus, whenever the singular values are not needed, the QR decomposition is preferred over the SVD. Figure 2.4 shows a tensor network diagram of the SVD (2.8).
An important property of the SVD is that it can be used to approximate a matrix A by a matrix $\tilde{A}$ of lower rank $\chi < \min(m, n)$. This *truncated SVD* can be performed by keeping only the largest $\chi < k$ singular values and omitting the corresponding columns of U and V:

$$A \approx \tilde{A} = \tilde{U}\tilde{S}\tilde{V},$$

with isometries $\tilde{U} \in \mathbb{C}^{n\times\chi}$, $\tilde{V} \in \mathbb{C}^{m\times\chi}$ and real diagonal matrix $\tilde{S} \in \mathbb{C}^{\chi\times\chi}$. It can be shown [3] that the truncated SVD minimizes the distance $\|A - \tilde{A}\|_\mathrm{F}$ between A and $\tilde{A}$ under the constraint $\mathrm{rank}(\tilde{A}) = \chi$. The truncated SVD is frequently used in tensor network algorithms to truncate tensors to a maximum bond dimension χ_max.

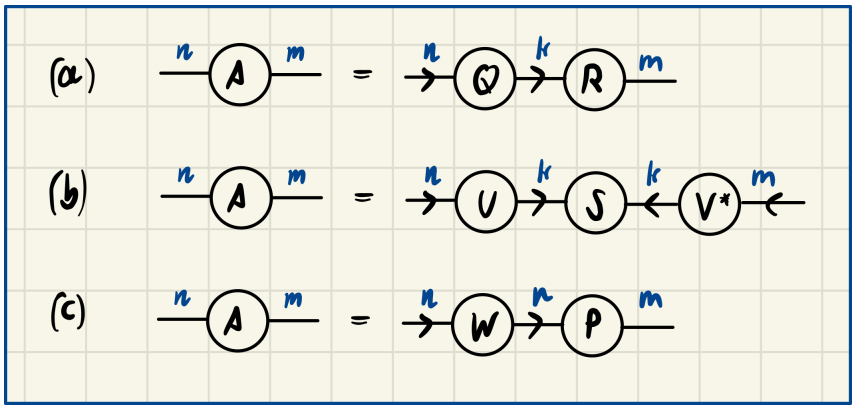


Figure 2.4: Different tensor decompositions are shown in tensor network diagram notation. The indices are decorated with bond dimensions. (a) QR decomposition (2.6). (b) Singular Value Decomposition (2.8)

2.3 Isometric Tensor Networks

An isometric tensor network is a tensor network whose diagrams bonds can be consistently assigned with arrows. In particular we will look at finite tensor networks where the arrows do not form any loops. In such networks, all arrows point to a single tensor, the *orthogonality center*. Such networks have the very useful property, that the error of local approximations around the orthogonality center can be computed locally (without contracting the full network). Let $\mathcal{N}$ be the tensor that is the result of contracting the full network, and let $\mathcal{M}$ be the tensor resulting from the contraction of a subregion of the network around the orthogonality center, where all arrows in the tensor network diagram point towards $\mathcal{M}$ (see figure 2.5(a) for an example in tensor diagram notation). Let us now approximate the sub-network $\mathcal{M}$ by a different sub-network $\mathcal{M}'$, which changes the contraction of the full network to $\mathcal{N}'$ (see 2.5(b)). We can compute the error ε of this approximation as

$$\begin{aligned}
\varepsilon^2 &= \|\mathcal{N} - \mathcal{N}'\|_{\mathrm{F}}^2 \\
&= \langle \mathcal{N} - \mathcal{N}', \mathcal{N} - \mathcal{N}' \rangle_{\mathrm{F}} \\
&= \|\mathcal{N}\|_{\mathrm{F}}^2 + \|\mathcal{N}'\|_{\mathrm{F}}^2 - 2\,\mathrm{Re}\,\langle \mathcal{N}, \mathcal{N}' \rangle_{\mathrm{F}} \\
&= \|\mathcal{M}\|_{\mathrm{F}}^2 + \|\mathcal{M}'\|_{\mathrm{F}}^2 - 2\,\mathrm{Re}\,\langle \mathcal{M}, \mathcal{M}' \rangle_{\mathrm{F}} \\
&= \|\mathcal{M} - \mathcal{M}'\|_{\mathrm{F}}^2,
\end{aligned}$$

where in the fourth step we used the fact that all tensors outside of the sub-network satisfy the isometry condition. As an example, the contraction of $\mathrm{Tr}\left(\mathcal{N}\mathcal{N}'^{\dagger}\right)$ is shown in figure 2.5(c). As one can see, the computation of the error reduces to a contraction of the local sub-networks. This greatly simplifies the computation of

optimal approximations of tensors especially for large networks, because the full network doesn't need to be contracted. As will become clear in the next section, when the tensor network represents a quantum state, this also makes it very easy to compute local expectation values, because the computation of the overlap of the wavefunction can be simplified to a contraction of a local environment around the orthogonality center. Additionally, approximations made by the truncated SVD 2.2 and the isometrization using the polar decomposition **??** are, when performed at the orthogonality center, globally optimal for isometric tensor networks, instead of only locally optimal for non-isometric tensor networks.

2.4 Matrix Product States (MPS)

The Density Matrix Renormalization Group (DMRG) algorithm, which was subsequently understood as a variational method over the class of Matrix Product States (MPS), has developed to be the de-facto standard for the numerical simulation of one-dimensional quantum systems. The success of this method is due to the remarkable ability of MPS to capture the area-law entanglement characteristics of ground states of gapped Hamiltonians. Additionally, due to the elegant diagrammatic notation for tensor networks, new algorithms can be developed and discussed efficiently and intuitively. Applications of MPS include finding ground and thermal states, real and imaginary time evolution, and the computation of dynamical properties of lattice Hamiltonians. In the following we give a brief introduction to MPS, for a more in-depth discussion see [4–6].
The state of a quantum many-body system can be written as

$$|\Psi\rangle = \sum_{i_1=1}^{d_1} \sum_{i_2=1}^{d_2} \cdots \sum_{i_N=1}^{d_N} \Psi_{i_1 i_2 \ldots i_N} |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_N\rangle \,.$$

where N is the number of subsystems (e.g. lattice sites or particles), and $\{|i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_N\rangle\}$, $i_j = 0, \ldots, d_j$ is a set of basis vectors of the full many-body Hilbert space

$$\mathcal{H} = \bigotimes_{j=1}^{N} \mathcal{H}_j,$$

with $\dim(\mathcal{H}_j) = d_j$ the dimension of the local Hilbert space of subsystem j. To simplify the notation, we will assume that the dimension of all local subsystems is the same, $d_j = d$. The d^N complex numbers $\Psi_{i_1 i_2 \ldots i_N}$ fully describe the quantum many-body state, and one can think of $\Psi \in \mathbb{C}^{d \times \cdots \times d}$ as a tensor of rank N. However, due to the size of the tensor scaling exponentially with system size, only very small system sizes are accessible computationally. One proceeds by writing Ψ as a tensor

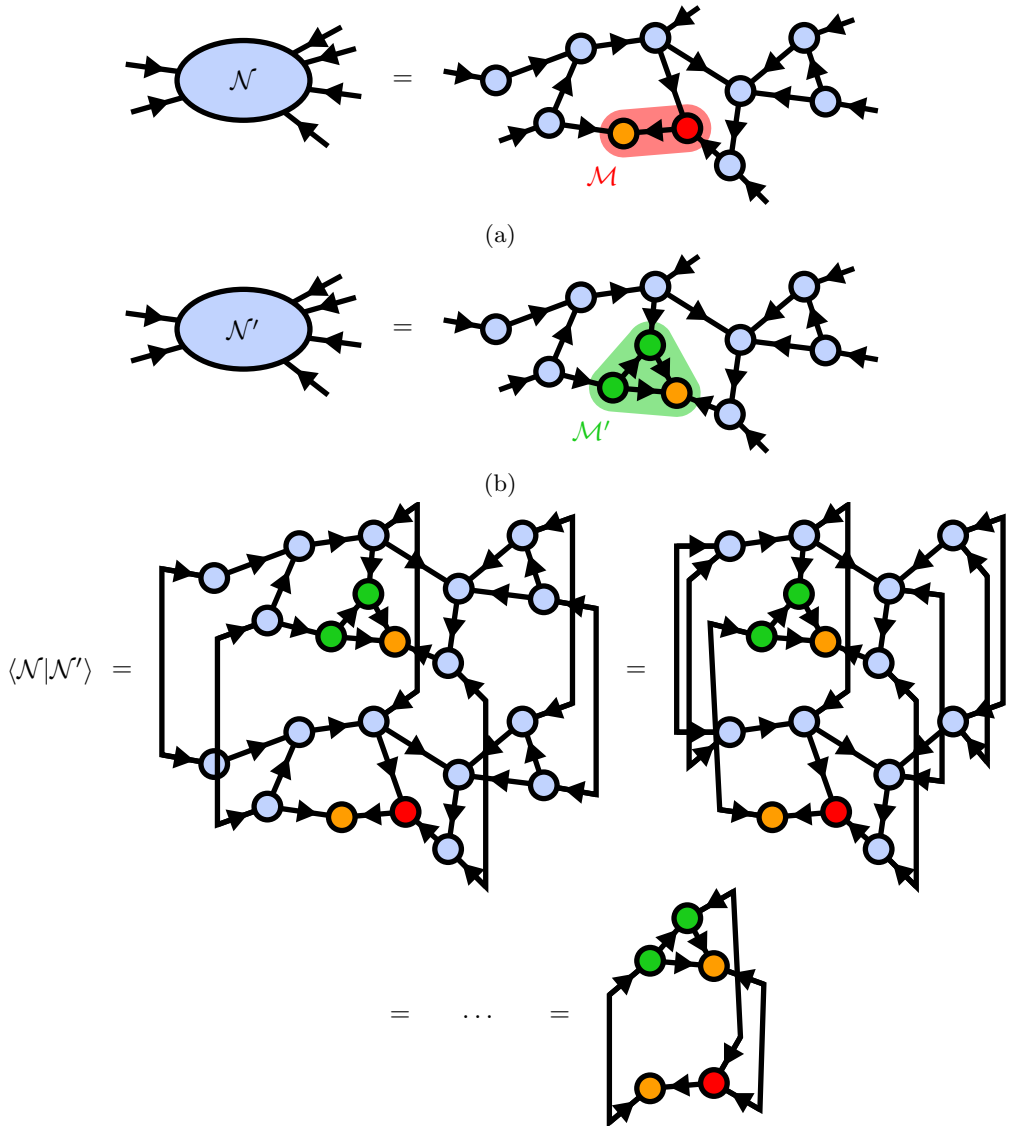


Figure 2.5: (a) An isometric tensor network $\mathcal{N}$ with the orthogonality center depicted in orange. The sub-network $\mathcal{M}$ is made up of all tensors in the red region. (b) The isometric tensor network $\mathcal{N}'$ with an updated sub-network $\mathcal{M}'$. (c) The computation of the trace $\mathrm{Tr}\,(\mathcal{N}\mathcal{N}')$ reduces to a contraction of the subregions $\mathrm{Tr}\,(\mathcal{M}\mathcal{M}')$ because of the isometry condition.

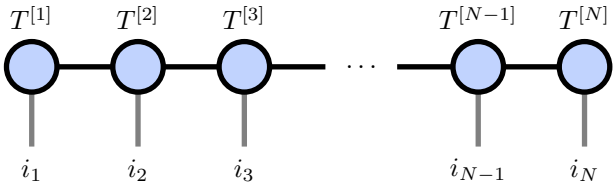


Figure 2.6: Diagrammatic representation of the Matrix Product State 2.9.

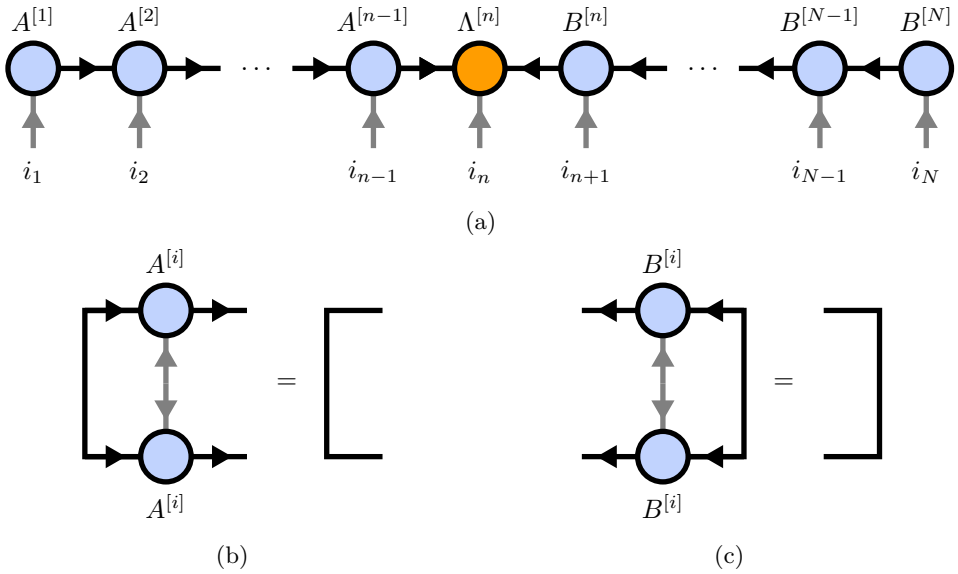


Figure 2.7: (a) Diagrammatic representation of an MPS in canonical form. (b) The left isometry condition. (c) The right isometry condition.

network of smaller tensors. A *Matrix Product State* (MPS) is constructed by introducing N rank-3 tensors $A^{[n]} \in \mathbb{C}^{d\times\chi_{n-1}\times\chi_n}$ and contracting them in a chain as

$$\Psi_{i_1 i_2 \cdots i_N} \coloneqq \sum_{\alpha_1=1}^{\chi_1} \sum_{\alpha_2=1}^{\chi_2} \cdots \sum_{\alpha_{N-1}=1}^{\chi_{N-1}} A^{[1],i_1}_{1,\alpha_1} A^{[1],i_2}_{\alpha_1,\alpha_2} \cdots A^{[N],i_N}_{\alpha_{N-1},1}, \tag{2.9}$$

where we have written the physical indices i_n as superscripts, such that the sums are performed only over subscripts. Note that in this notation the bond dimensions at the two ends of the chain is $\chi_0 = \chi_N = 1$, and we can interpret the tensors $A^{[1]}$ and $A^{[N]}$ as tensors of rank-2. A tensor diagram of the MPS (2.9) is given in figure 2.6. An important property of MPS is the existence of a *canonical form* as an isometric tensor network, where a single tensor $A^{[n]}$ is selected as the orthogonality center. One can bring an arbitrary MPS into this canonical form through successive

QR-decompositions or SVDs, starting at the outer ends of the chain and isometrizing one tensor at a time, until the orthogonality center is reached [4]. In figure 2.7a an MPS in canonical with the orthogonality center at subsystem n is visualized in diagrammatic notation. The canonical form greatly simplifies many operations on MPS and allows for the formulation of efficient algorithms, where many contractions reduce to identity due to the isometry condition (2.5), see also figure 2.7b and 2.7c. For example, the expectation value $\langle\Psi|\,\hat{O}\,|\Psi\rangle$ of a one-site operator $\hat{O} \in \mathbb{C}^{d\times d}$ acting on site n can for a general MPS be computed as

$$\begin{aligned}\langle\Psi|\,\hat{O}\,|\Psi\rangle &= \sum_{i_1,\ldots,i_N,j_n=1}^{d} \Psi_{i_1,i_2,\ldots,i_N}\Psi^*_{i_1,\ldots,i_{n-1},j_n,i_{n+1},\ldots,i_N}\,\langle j_n|\,\hat{O}\,|i_n\rangle \\ &= \sum_{i_1,\ldots,i_N,j_n=1}^{d} \left(A^{[1],i_1}\cdots A^{[N],i_N}\right) \\ &\qquad\cdot \left(A^{[1],i_1*}\cdots A^{[N],j_n*}\cdots A^{[N],i_N*}\right)\cdot \hat{O}_{i_n,j_n},\end{aligned} \tag{2.10}$$

where the $A^{[n],i_n}$ are interpreted as matrices for $1 < n < N$ and as row/column vectors for $n = 1, N$ such that the product

$$\left(A^{[1],i_1}\cdots A^{[N],i_N}\right)$$

gives a scalar. The contraction (2.10) is visualized as a tensor diagram in figure 2.8. Here, the advantage of the diagrammatic notation becomes appearant: It is much easier to understand how tensors are contracted when expressing the contraction in terms of tensor network diagrams. The computational cost of computing the expectation value like this scales linear with the system size $\mathcal{O}\left(N\chi^3 d\right)$, where χ is the maximum virtual bond dimension $\chi = \max\{\chi_1,\ldots,\chi_N\}$. If the MPS is however given in canonical form with the orthogonality center at site n, the computation reduces to a contraction of only three tensors as can be seen in figure 2.9, and the computational cost $\mathcal{O}\left(\chi^3 d\right)$ becomes independent of system size.
Until now, the MPS representation of $|\Psi\rangle$ is still exact. One can approximate a MPS by restricting the virtual bond dimension to a maximal bond dimension $\chi_n < \chi_{\max}$. In this case, the number of parameters that need to be stored to describe the state is reduced from $\mathcal{O}\left(d^N\right)$ to $\mathcal{O}\left(N\chi_{\max}^2 d\right)$. To arrive at this approximation, two neighbouring tensors can be contracted and split via a truncated SVD, keeping only the $\chi_{\max}$ largest singular values. If the orthogonality center of the MPS is at one of the two tensors, this approximation is globally optimal as explained in section 2.3. Additionally, this SVD at the orthogonality center is related to the Schimdt decomposition of a bipartite system

$$|\Psi\rangle = \sum_{\alpha=1}^{\chi_n} \lambda_\alpha \left|\Psi_\alpha^{[L]}\right\rangle \otimes \left|\Psi_\alpha^{[R]}\right\rangle,$$

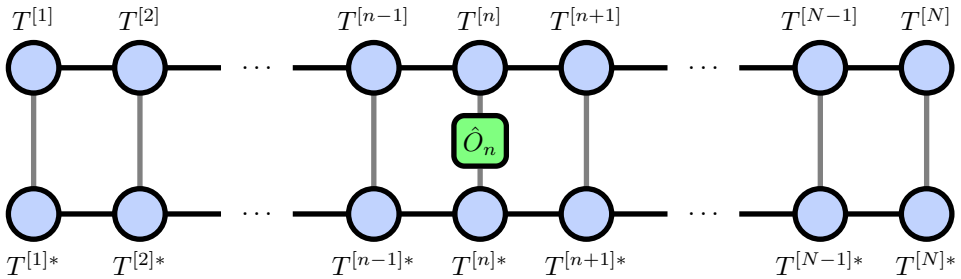


Figure 2.8: The computation of the expectation value of a local operator can be computed by contracting the MPS with its conjugate transpose, with the operator "sandwiched" between.

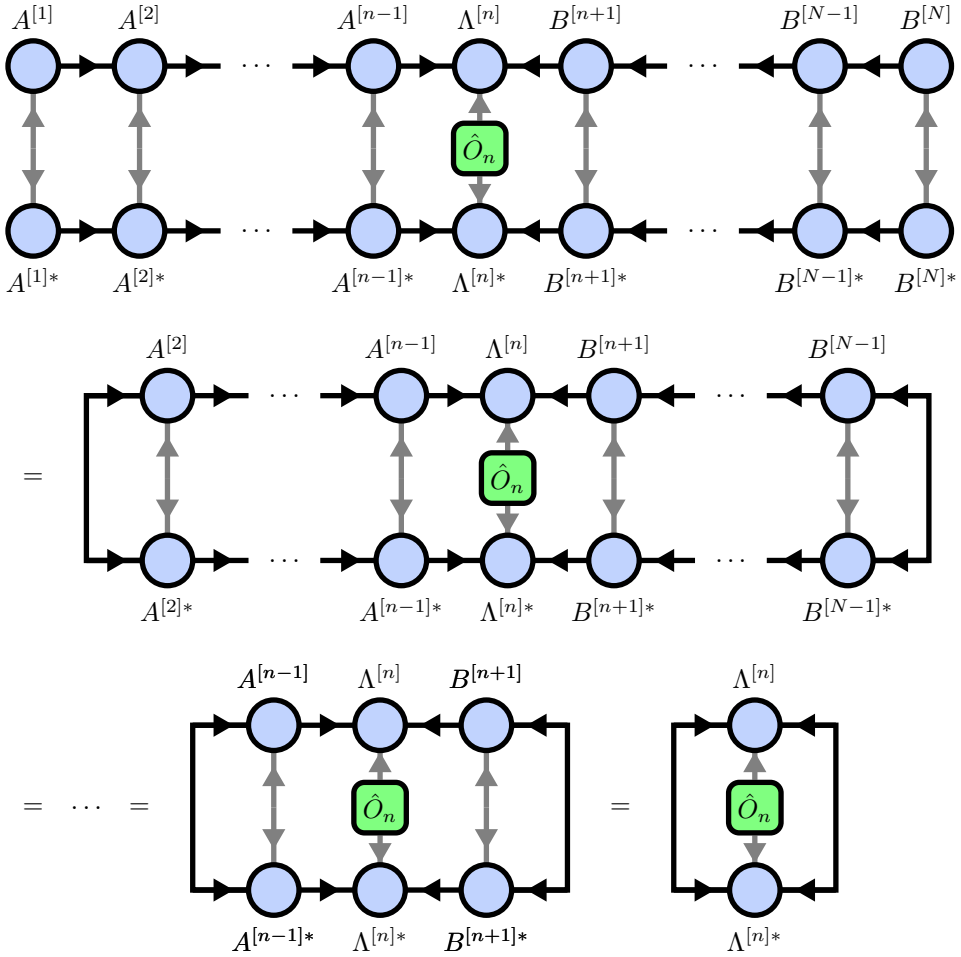


Figure 2.9: If the MPS is in canonical form, the computation of the expectation value of a local operator can be simplified to a contraction of three tensors using the isometry condition.

where the chain is split into a left and right subsystem with orthogonal basis vectors $\left|\Psi_\alpha^{[L]}\right\rangle$ and $\left|\Psi_\alpha^{[R]}\right\rangle$ as visualized in figure **??**. In this case, the Schmidt values $\lambda_\alpha >= 0$ coincide with the singular values! One can further use this to compute the Von-Neumann entanglement entropy

$$S = -\sum_{\alpha=1}^{\chi_n} \lambda_\alpha^2 \log\left(\lambda_\alpha^2\right),$$

quantifying the amount of entanglement between the left and right subsystems. If the state is normalized, it additionally holds

$$\sum_{\alpha=1}^{\chi_n} \lambda_\alpha^2 = 1.$$

Thus, how well an MPS of a given bond dimension $\chi_{\max}$ is able to represent a given quantum state is highly dependent on the Schmidt spectrum $\{\lambda_\alpha\}$ at the different bipartitions of the chain. If the Schmidt values decrease exponentially, only an exponentially small part of the entanglement structure is truncated and the truncated MPS is a good approximation for the original state. It can be shown [7, 8] that for ground states of local, gapped, one dimensional Hamiltonians there holds an *area law*: The entanglement entropy at arbitrary bipartitions of the chain is bounded by a constant

$$S \leq S_{\max},$$

where $S_{\max}$ is independent of the system size. This is in contrast to the fact that the entanglement of states drawn randomly from the many-body Hilbert space on average exhibits *volume law* scaling

$$\mathbb{E}\left[S\right] > \min\left(N_L, N_R\right)\log(d),$$

where N_L and N_R are the number of subsystems in the left and right bipartition. Hence, ground states of gapped Hamiltonians are very nongeneric. Note that the constant $S_{\max}$ scales with the correlation length of the system, which diverges when approaching critical points.
It is immediately clear that truncated MPS by construction exhibit area law entanglement scaling, if the local subsystems that are represented by each tensor correspond to physical systems on a 1D chain. The maximal entanglement entropy for a bipartition can be reached when all Schmidt values are equal, $\lambda_\alpha = 1/\sqrt{\chi_n}$, and thus

$$S \leq \log\left(\chi_{\max}\right)$$

for arbitrary bipartitions of the chain. One can conclude that MPS are good approximations for ground states of gapped 1D Hamiltonians away from criticality.

For completeness we note that the truncation of all bonds of an MPS is a highly non-linear optimization problem and the naive algorithm of truncating each bond with an SVD does in general not lead to a minimal error. A variational compression procedure can often be used to obtain a lower error at the same maximum bond dimension $\chi_{\max}$ [4].
Many algorithms have been formulated in the language of MPS. Most notably, the Density Matrix Renormalization Group (DMRG) algorithm can be used for finding ground states of local lattice Hamiltonians [4]. Time evolution of MPS can be performed with the the Time Evolving Block Decimation (TEBD) algorithm [9, 10] or the Time Dependant Variational Principle (TDVP) [11, 12]. In the following, we will briefly discuss TEBD, as the algorithm can be generalized easily to isometric tensor product states of higher dimension, which we will do in section **??**
Assume that we are given a quantum state $|\Psi\rangle$ in the form of an MPS and a Hamiltonian $\hat{H}$ that can be written as a sum of nearest-neighbour operators $\hat{h}_{j,j+1}$,

$$\hat{H} = \sum_{j=1}^{N-1} \hat{h}^{[j,j+1]}.$$

The state $|\Psi\rangle$ can be evolved in time as

$$|\Psi(t)\rangle = \hat{U}(t)\,|\Psi\rangle = e^{-it\hat{H}}\,|\Psi\rangle\,,$$

where we set $\hbar = 1$. The time evolution operator $U(t)$ is in general very hard to compute and handle exactly. Thus, $U(t)$ is approximated using a Suzuki-Trotter decomposition. We start by decomposing the time evolution into a series of K small time steps $\Delta t = t/K$ as

$$U(t) = e^{-it\hat{H}} = \left(e^{-i\Delta t\hat{H}}\right)^K = (U(\Delta t))^K\,.$$

Next, we split the Hamiltonian into terms acting on even and odd bonds

$$\hat{H} = \sum_{j \text{ even}} \hat{h}^{[j,j+1]} + \sum_{j \text{ odd}} \hat{h}^{[j,j+1]} =: H_{\text{even}} + H_{\text{odd}}.$$

Using the Zassenhaus formula

$$e^{\varepsilon(A+B)} = e^{\varepsilon A} e^{\varepsilon B} e^{-\frac{\varepsilon^2}{2}[A,B]} e^{\frac{\varepsilon^3}{6}(2[B,[A,B]]+[A,[A,B]])} \dots$$

which can be derived from the Baker-Campbell-Hausdorff formula, we can approximate

$$\begin{aligned} U(\Delta t) &= e^{-i\Delta t(H_{\text{even}}+H_{\text{odd}})} = e^{-i\Delta t H_{\text{even}}} e^{-i\Delta t H_{\text{odd}}} + \mathcal{O}\left(\Delta t^2\right) \\ &= \hat{U}^{\text{TEBD1}}(\Delta t) + \mathcal{O}\left(\Delta t^2\right), \end{aligned}$$

which is called a Suzuki-Trotter decomposition of first order. Since operators acting on even bonds commute with each other, the exponentials $e^{-i\Delta t H_{\text{even}}}$ factorize,

$$e^{-i\Delta t H_{\text{even}}} = e^{-i\Delta t \sum_{j \text{ even}} h^{[j,j+1]}} = \prod_{j \text{ even}} e^{-i\Delta t h^{[j,j+1]}},$$

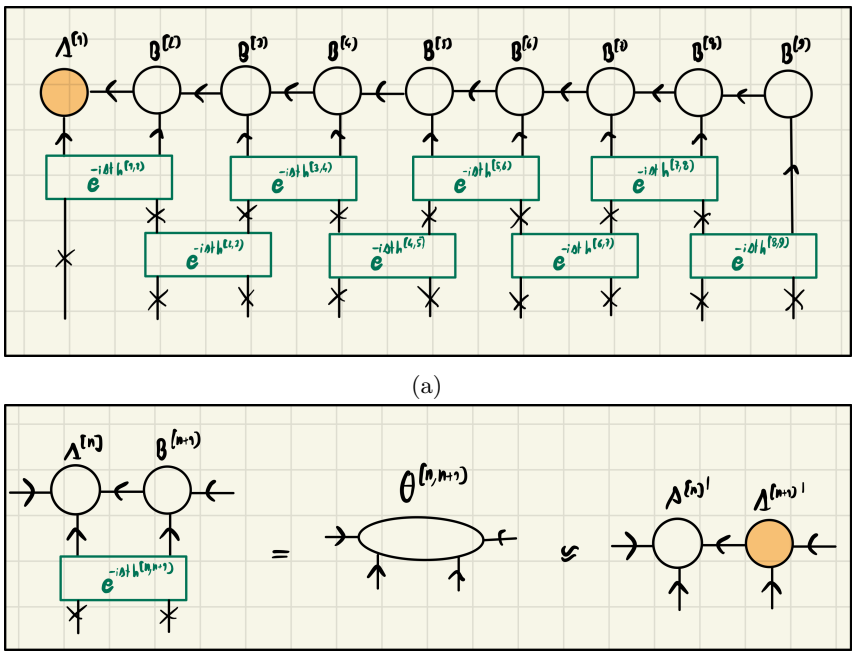
and the same is true for the exponentials $e^{-i\Delta t H_{\text{odd}}}$. Each bond operator $e^{-i\Delta t h^{[j,j+1]}}$ acting on the combined Hilbert space of sites j and $j+1$ can be reshaped into a tensor of rank 4. The application of the operator $\hat{U}^{\text{TEBD1}}(\Delta t)$ to a state in MPS form can thus be written as the tensor network in figure 2.10a. To perform a single TEBD iteration corresponding to a time evolution of Δt we want to approximate this tensor network by a new MPS. This can be done by moving the orthogonality center from left to right, applying the bond operators $e^{-i\Delta t h^{[j,j+1]}}$ while keeping the MPS structure. The process of applying a single bond operator is shown in figure 2.10b. First, the two site tensors and $e^{-i\Delta t h^{[j,j+1]}}$ are contracted into a single tensor, that is then split and truncated again using an SVD. By sweeping twice across the MPS, first applying the bond operators on all even bonds and then the bond operators on all odd bonds, we perform a full TEBD iteration. There exist two sources of errors, the truncation error of the truncated SVD and the error of the Suzuki-Trotter decomposition. The truncation error can be controlled by choosing a larger bond dimension χ, allowing the representation of more entanglement and thus the evolution to larger times. However, generally the amount of entanglement grows exponentially in time, necessitating an exponentially growing bond dimension and practically limiting the algorithm to small times. A smaller Suzuki-Trotter error can be achieved by choosing smaller time steps Δt or by performing a higher-order Suzuki-Trotter decomposition. For example, a second order decomposition can be computed as

$$\begin{aligned} U(\Delta t) &= e^{-i\Delta t(H_{\text{even}}+H_{\text{odd}})} = e^{-i\frac{\Delta t}{2}H_{\text{even}}} e^{-i\Delta t H_{\text{odd}}} e^{-i\frac{\Delta t}{2}H_{\text{even}}} + \mathcal{O}\left(\Delta t^3\right) \\ &= \hat{U}^{\text{TEBD2}}(\Delta t) + \mathcal{O}\left(\Delta t^3\right). \end{aligned}$$

and can be applied to a MPS similarly as the first order decomposition. For higher order Suzuki-Trotter decompositions see [13].

2.5 Isometric Tensor Product States in 2D

The natural generalization of MPS to higher dimensional lattices is given by *Projected Entangled Pair States* (PEPS). A PEPS is constructed similar to a MPS by representing the local subsystem on each lattice site i with the index σ_i of a tensor $T_i^{\sigma_i}$ and connecting nearest-neighbour tensors with virtual bonds. The



(a)

(b)

Figure 2.10: TODO

quantum state can then be written as

$$|\Psi\rangle = \sum_{\sigma_1,\sigma_2,\ldots,\sigma_N} \mathcal{C}\left(T_1^{\sigma_1}, T_2^{\sigma_2}, \ldots, T_N^{\sigma_N}\right) |\sigma_1, \sigma_2, \ldots, \sigma_N\rangle \,,$$

where $\mathcal{C}(\ldots)$ denotes the contraction of the full network along all virtual bonds. As an example, we draw a PEPS on a square lattice in figure 2.11a.
PEPS are able to efficiently represent area-law states in two and higher dimensions [5]. Remarkably, PEPS can even handle correlations decaying polynomially with separation distance [14], whereas MPS can only handle exponentially decaying correlations. Polynomially decaying correlations are characteristic for critical points. Unfortunately, it is not generally possible to bring a PEPS into an exact canonical form due to the presence of closed loops. Thus, already the computation of local expectation values scales exponentially with system size and can in practice be only done approximately, e.g. using the boundary MPS method [5] or corner transfer matrices [15]. Moreover, algorithms for ground state search and time evolution have computational costs scaling with high powers of the bond dimension. For example the cost of a full update TEBD or DMRG iteration is dominated by the contraction of an effective environment, scaling as $\mathcal{O}\left(D^{10}\right)$ [16].

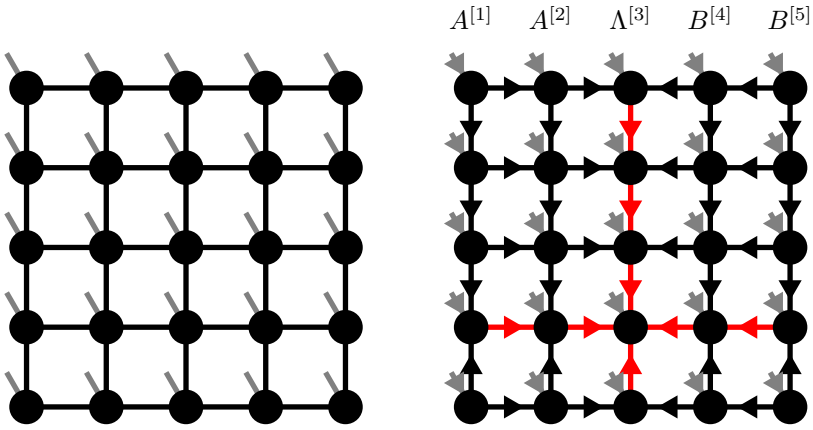
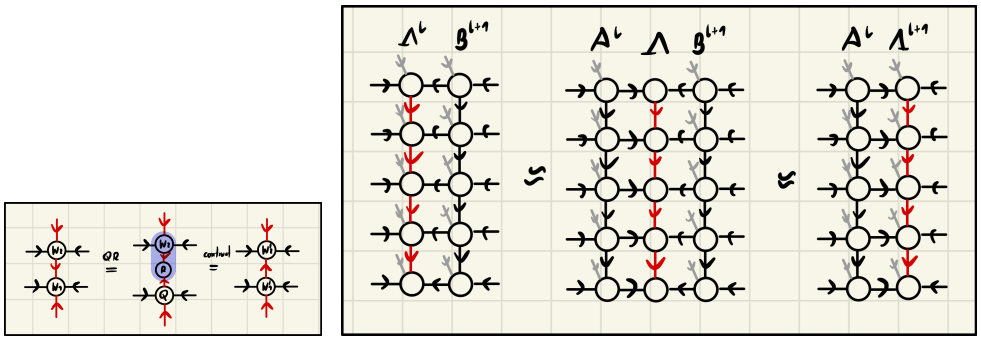


Figure 2.11: Tensor Networks representing two dimensional states on a square lattice. (a) A Projected Entangled Pair State (PEPS). (b) An isometric Tensor Product State (isoTPS).

Recently, the new class of *isometric Tensor Product States* (isoTPS) has been introduced [1, 17, 18], generalizing the canonical form of MPS to higher dimensions by enforcing isometry constraints. This allows for efficient computation of local expectation values and greatly lowers the cost of some algorithms. The downside to this approach is that the set of states representable by isoTPS is smaller than the set of state representable by PEPS. It is thus an interesting question to ask which kinds of states and, more generally, which kinds of quantum phases can still be represented by isoTPS.
In the following, we will give a brief introduction to the isoTPS defined in [1] and discuss their properties. A two-dimensional isoTPS on the square lattice is constructed by enforcing the isometry conditions shown in figure 2.11b. All isometries are chosen in such a way that all arrows point towards a special row and column, called the *orthogonality hypersurface* of the isoTPS. The term "hypersurface" is chosen in anticipation of a generalization to higher dimensions. Because of the isometry condition, one can think of the contractions of each of the four regions outside the orthogonality hypersurface as orthogonal boundary maps [2]. The single tensor with only incoming arrows is called the *orthogonality center*. local expectation values of operators acting in the vicinity of the orthogonality center can be computed efficiently because most contractions reduce to identity, similar to the computation of local expectation values in MPS (see figure 2.9). The orthogonality center can be moved along the orthogonality hypersurface simply and exactly using a QR-decomposition as shown in figure 2.12a.
Moving the orthogonality surface is a harder problem, which in general can only be



(a) (b)

Figure 2.12: Moving the orthogonality center and the orthogonality hyper surface around is a central problem in isoTPS applications. (a) Moving the orthogonality center along the orthogonality hypersurface can be done easily via QR-decompositions. (b) An orthogonality column can be moved by first solving equation (2.12) variationally and then absorbing Λ into B_{l+1} via the standard MPO-MPS multiplication and MPS compression algorithms.

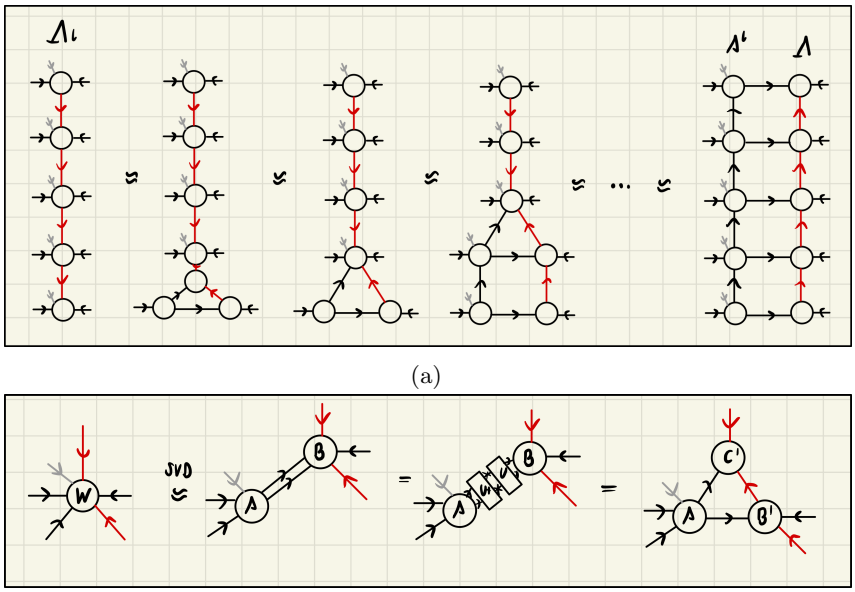
done approximately. In analogy to MPS and as shown in figure 2.11b, we call columns left of the orthogonality hypersurface A_l and tensors right of the orthogonality hypersurface B_l, with $l = 1, 2, \ldots, L$ and L the linear system size. Moving the orthogonality hypersurface Λ_l one column to the right can be expressed as solving the problem

$$\Lambda_l B_{l+1} = A_l \Lambda_{l+1}, \tag{2.11}$$

where the notation $\Lambda_l B_{l+1}$ means the contraction of columns Λ_l and B_{l+1} along their connecting bonds. Instead of (2.11), one can solve the simpler auxillary problem

$$\Lambda_l = A^l \Lambda, \tag{2.12}$$

where Λ is a column of tensors with no physical indices, as shown in figure 2.12b. This column can then be absorbed into B_{l+1} via the standard algorithm of applying an MPO to an MPS and subsequent MPS compression [4]. One can variationally solve problem (2.12) by minimizing the distance $|\Lambda_l - A_l \Lambda|$, sweeping back and forth through the tensors while respecting the isometry condition. It is however found in [1] that a single unzipping sweep, called the *Moses Move* (MM), provides a solution very close to the variational one whilst being far quicker. The MM can also be used as a good initialization for the variational algorithm. We sketch the MM in figure 2.13a. We start from the bottom of the orthogonality hypersurface column and split the tensors one after the other using a tripartite decomposition. A single tripartite decomposition of a tensor W is shown in figure 2.13b. First, W is split into two tensors A and B via a truncated SVD $W = U(SV) = AB$ with A an isometry. The bond connecting A and B is reshaped into two bonds. Next, it is



(a)

(b)

Figure 2.13: (a) The Moses Move (MM) splits the column Λ_l into A_l and Λ via a single unzipping sweep of tripartite decompositions. (b) Tripartite decomposition of the tensor W as explained in the text.

important to note that the full contraction is invariant under the insertion of a unitary and its conjugate transpose, $AB = (AU^\dagger)(UB)$, with $(AU^\dagger)$ still satisfying the isometry condition. This degree of freedom can be used to *disentangle* the tensor B along the direction of the red bonds. Accordingly, we choose U such that the truncation error or some entanglement measure is minimized for splits along the direction of the red bonds. Choosing a good disentangling unitary is crucial for a successful tripartite decomposition and will be discussed further in section **??**. Assume for now that a good disentangling unitary has been found. After contracting $(AU^\dagger)$ and (UB), a truncated SVD is used to split (UB) into tensors B' and C' as shown in figure 2.13b, completing the tripartite decomposition.
Because the orthogonality center can be moved easily along the orthogonality hypersurface, one can think of the orthogonality hypersurface along a column or row as a 1D MPS with an enlarged physical bond dimension grouping together the phsical and the two ancilla legs pertruding from the orthogonality hypersurface tensors. Standard MPS algorithms can then be generalized to isoTPS by performing one iteration of the algorithm on the orthogonality hypersurface MPS, before moving the hypersurface via MM or variational optimization and repeating the

procedure. As an example, we will discuss TEBD^2, the generalization of $TEBD$ to an isoTPS on a 2D square lattice.

isoTPS are a new class of states that enable the implementation of faster algorithms for e.g. ground state search and time evolution, compared to PEPS. The expressional power of isoTPS has been studied in [19], where it was found that isoTPS with finite bond dimension can exactly represent ground state wavefunctions of string-net liquid models, showing that long-range entanglement does not form an obstruction for isoTPS representations and suggesting that the ground states of gapped Hamiltonians with gappable edges can be efficiently represented as an isoTPS. There have also been works discussing the computational complexity of isoTNS [20] and relating isoTNS to quantum circuits [21, 22]. In [23], topological phase transitions were studied with isoTPS, showing that isoTPS can represent some critical states with power-law correlations. A DMRG^2-algorithm was implemented on isoTPS in [2] and used to compute dynamical structure factors of ground states using real time evolution. IsoTPS were also extended to fermionic systems [24], to two dimensional strips of infinite length [25], and to three dimensional cubic lattices [26]. They have also been used to compute properties of two dimensional thermal states [27].

Chapter 3

diagonal isometric Tensor Product States (disoDTPS)

In the following, we will introduce a new class of isometric tensor tensor states which we call *diagonal isometric tensor product states* (disoTPS). This class of tensor product states is in many ways similar to the isometric tensor product states discussed in section 2.5, with some important differences.

3.1 Network Structure

The structure of a disoTPS on a square lattice is shown in figure 3.1. It can be constructed in three steps. First, a square PEPS is rotated by 45°. Next, the orthogonality hypersurface is constructed as a column of auxillary tensors. The auxillary tensors are connected in a line similar to an MPS and placed between between two columns of PEPS tensors. Note that, in contrast to the standard isoTPS the tensors of the orthogonality hypersurface do not carry any physical degrees of freedom and are only connected via virtual bonds to the neighboring tensors. Lastly, the isometry condition is enforced such that all arrows point towards the orthogonality hypersurface. Tensors left of the orthogonality surface are thus brought into a left-isometric form and tensors right of the orthogonality surface are brought into a right-isometric form, as shown in figure 3.1. The auxillary tensors making up the orthogonality hypersurface are isometrized such that all arrows point towards a single auxillary tensor, the orthogonality center.
In the following, we will denote the auxillary tensors by W_i, and the tensors carrying physical degrees of freedom with T_i. The bonds connecting two T-tensors or a T-tensor and a W-tensor are truncated to a maximal bond dimension of D, while the maximal bond dimension between two W-tensors is denoted as χ. In practice it is found that setting $\chi = f \cdot D$ with an integer $f \geq 1$ produces good results.
Similar to the standard isoTPS, the orthogonality center can easily and exactly be moved along the orthogonality hypersurface using QR-decompositions. Moving the orthogonality hypersurface to the left or to the right is a harder problem and will be discussed in section 3.2.

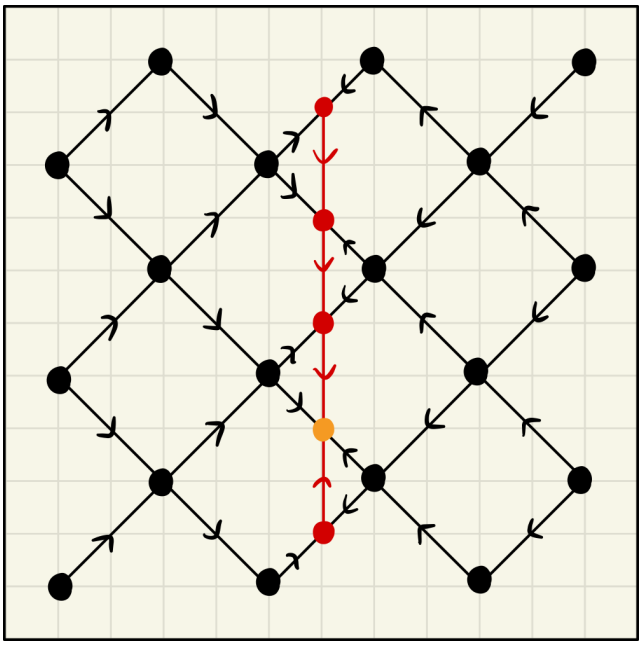


Figure 3.1: A diagonal isometric tensor network on a square lattice is constructed from site tensors T_i (drawn in black) and an orthogonality hypersurface of auxillary tensors W_i (drawn in red). The orthogonality hypersurface is rotated by 45° with respect to the lattice.

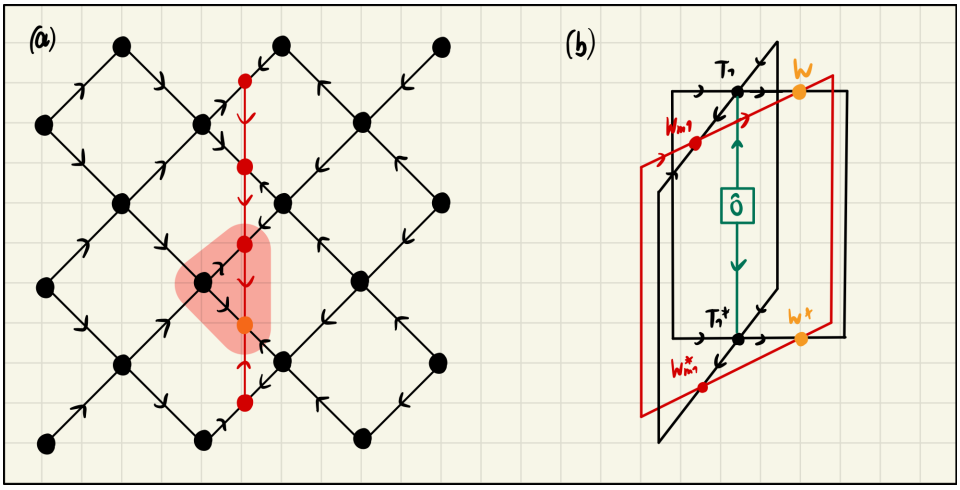


Figure 3.2: (a) The one-site wavefunction around a site i is the sub network containing the site tensor T_i and the two connected auxillary tensors. (b) Computation of a single site expectation value reduces to the shown contraction over a one-site wavefunction, its complex conjugate, and the one-site operator $\hat{O}_i$.

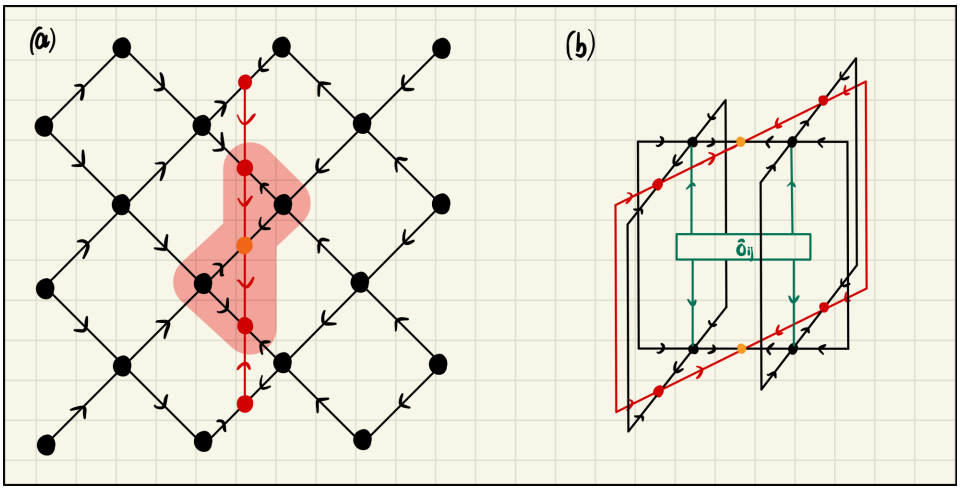


Figure 3.3: (a) The two-site wavefunction around neighboring sites i and j is the sub network containing the site tensors T_i and T_j and the three connected auxillary tensors. (b) Computation of a two-site expectation value reduces to the shown contraction over a two-site wavefunction, its complex conjugate, and the two-site operator $\hat{O}_{ij}$.

As in MPS and standard isoTPS, disoTPS allow for the fast computation of expectation values of local operators. The expectation value $\langle\Psi|\,\hat{O}_i\,|\Psi\rangle$ of a one-site operator $\hat{O}_i$ acting on site i can be computed as follows: First, the orthogonality center is moved next to site i. We next define the *one-site wavefunction* as the sub-network containing the site tensor T_i and the two connected W-tensors. Note that the one-site wavefunction is connected to its environment only by bonds with incoming arrows. Next the wavefunction is contracted with its complex conjugate, sandwhiching the operator $\hat{O}_i$ between the two. Due to the isometry condition, this reduces to a contraction of only the one-site wavefunction, its complex conjugate, and the operator $\hat{O}_i$, as shown in figure 3.2. This contraction has a computational cost scaling as $\mathcal{O}\left(\chi^3 D^3 + D^6 d^2\right)$ and gives as result the desired expectation value. The expectation value $\langle\Psi|\,\hat{O}_{ij}\,|\Psi\rangle$ of a two-site operator $\hat{O}_{ij}$ acting on two neighbouring sites i and j, also called *bond operator*, can be computed similarly. First, the orthogonality center is moved such that it sits in the middle of the bond connecting sites i and j. The *two-site wavefunction* is then defined as the subnetwork containing the two site tensors T_i and T_j and three W-tensors as shown in figure 3.3, such that again all legs connecting the subnetwork to its environment are only decorated with arrows pointing towards the two-site wavefunction. The computation of the expectation value then reduces to the contraction of only the two-site wavefunction with its complex conjugate and the bond operator $\hat{O}_{ij}$. The computational cost of this contraction scales as $\mathcal{O}\left(\chi^3 D^3 d^2\right)$.

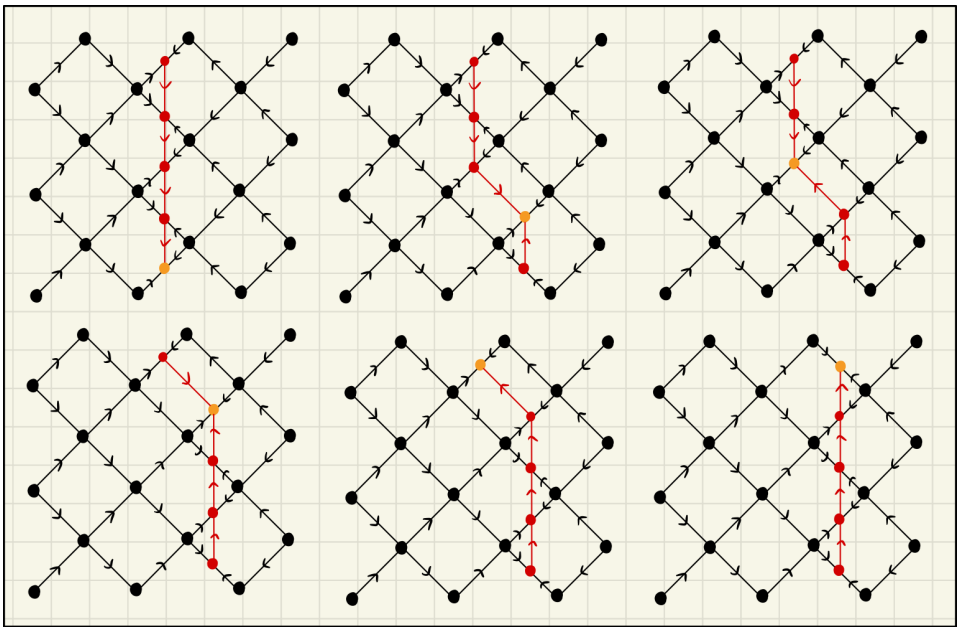


Figure 3.4: Two YB-moves are used to shift the orthogonality hypersurface one column to the right. In the last step, the orthogonality center can be moved across the T-tensor by contracting the two tensors and performing a QR-decomposition.

3.2 Yang-Baxter Move

Most algorithms implemented on disoTPS require an efficient procedure for moving the orthogonality surface, where the error introduced by this procedure should be as small as possible. For isoTPS, the current best procedure is given by the Moses Move, followed by an optional variational optimization.
In analogy to the MM we look for a procedure to iteratively shift the orthogonality surface through one column of T-tensors as shown in figure 3.4. A single iteration of this process is shown in figure 3.5. The two tensors W_1 and W_2, which are part of the orthogonality hypersurface, are "pulled through" the site tensor T, resulting in the updated tensors T', W_1' and W_2'. To keep the isometric structure of the network, T' and W_1' must be isometries, while W_2' must be a tensor of norm one (the new orthogonality center). Due to the visual similarity to the Yang-Baxter equation we call this procedure the *Yang-Baxter* (YB) move.
We denote the state represented by the disoTPS before the YB move by $|\Psi\rangle = |\Psi(W_1, W_2, T)\rangle$ and the state after the YB move by $|\Psi'\rangle = |\Psi'(W_1', W_2', T')\rangle$. One can think of the YB move as an optimization problem

$$\left(T'_{\text{opt}}, W'_{1,\text{opt}}, W'_{2,\text{opt}}\right) = \underset{T', W_1', W_2'}{\operatorname{argmin}} \left\| |\Psi\rangle - |\Psi'\rangle \right\|_{\text{F}} \tag{3.1}$$

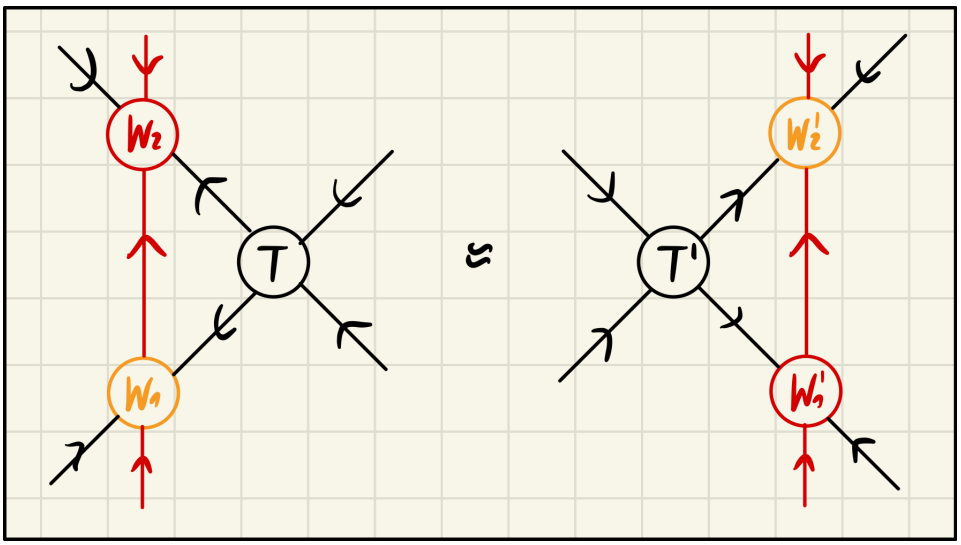


Figure 3.5: The Yang-Baxter (YB) move is the procedure of "pulling" two auxillary tensors W_1 and W_2 through a site tensor T.

under the constraints

$$T'^{\dagger}T' = \mathbb{1}, \quad W_1'^{\dagger}W_1' = \mathbb{1}, \quad \left\|W_2'\right\|_{\mathrm{F}} = 1. \tag{3.2}$$

We further rewrite the error of the YB move as

$$\begin{aligned}\left\||\Psi\rangle - \left|\Psi'\right\rangle\right\|_{\mathrm{F}} =& \sqrt{\langle\Psi|\Psi\rangle + \langle\Psi'|\Psi'\rangle - 2\,\mathrm{Re}\,\langle\Psi|\Psi'\rangle} \\ =& \sqrt{2 - 2\,\mathrm{Re}\,\langle\Psi|\Psi'\rangle},\end{aligned}$$

where in the second step we used the fact that the wave function is normalized to one, $\langle\Psi|\Psi\rangle = \langle\Psi'|\Psi'\rangle = 1$. It follows that the optimization problem of minimizing the error becomes the problem of maximizing the overlap

$$\left(T'_{\mathrm{opt}}, W'_{1,\mathrm{opt}}, W'_{2,\mathrm{opt}}\right) = \underset{T', W_1', W_2'}{\mathrm{argmax}}\ \mathrm{Re}\left\langle\Psi\middle|\Psi'\right\rangle \tag{3.3}$$

under the constraints (3.2). Because the only tensors that are changed by the YB move are W_1, W_2 and T and the three tensors make up a subregion of the full network with only incoming arrows, we can use the isometry condition and the computation of the overlap reduces to a contraction of only six tensors as shown in figure 3.6a.
In the following, we present two explicit algorithms for performing the YB move. The first algorithm (see section 3.2.1) is an iterative optimization using local updates respecting the constraints. The second algorithm (see section 3.2.2) is a tripartite decomposition with disentangling similar to the tripartite decomposition used in the MM. In section 3.2.3 we will compare the two algorithms.

3.2.1 Iterative optimization with local updates

To solve the constrained optimization problem (3.3) we proceed by maximizing the overlap while only varying the parameters of one of the three tensors T', W_1' or W_2', treating all other tensors as constant. For example, let us keep W_1' and W_2' fixed and optimize T'. We first contract all tensors except T' into an environment E as shown in figure 3.6(b). We can then write the optimization problem as

$$T' = \underset{T'^\dagger T=\mathbb{1}}{\operatorname{argmax}} \operatorname{Re} \left\langle \Psi \middle| \Psi' \right\rangle = \underset{T'^\dagger T=\mathbb{1}}{\operatorname{argmax}} \operatorname{Re} \left\langle T', E \right\rangle_{\mathrm{F}} = \underset{T'^\dagger T=\mathbb{1}}{\operatorname{argmax}} \operatorname{Re} \operatorname{Tr} \left(T'^\dagger E \right).$$

This problem is known as the *orthogonal Procrustes problem* and permits the closed form solution $T'_{\mathrm{opt}} = UV^\dagger$, where U and V are computed using the SVD $E = USV^\dagger$. For the derivation of this solution see appendix A.1. The tensors W_1' and W_2' can be optimized similarly. The full algorithm is then performed by sweeping over the three tensors, optimizing them iteratively until convergence. Tensor diagrams for the algorithm are shown in figure 3.6. We discuss one iteration of the algorithm in more detail:

1. We contract all tensors except T' into an environment E and perform an SVD $E = USV$. The tensor T' is then updated as $T' \leftarrow UV^\dagger$. See figure 3.6(b).
2. We contract all tensors except W_1' into an environment E and perform an SVD $E = USV$. The tensor W_1' is then updated as $W_1' \leftarrow UV^\dagger$. See figure 3.6(c).
3. We contract all tensors except W_2' into an environment E. The tensor W_1' is then updated as $W_1' \leftarrow E / \|E\|$. See figure 3.6(d).

These three steps are repeated until a termination criterion is met, for example until the decrease in error after one iteration is smaller than a given threshold or if a given maximum number of iterations is exceeded.

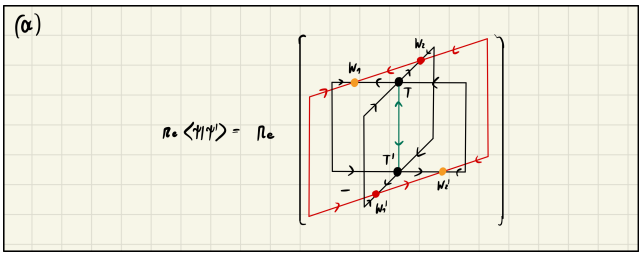
3.2.2 Tripartite decompositon using an SVD and disentangling

Alternatively, the constrained optimization problem (3.1) can be solved via two successive SVDs with an optional disentangling prodcedure with the goal of reducing the truncation error or some entanglement measure. This is the same algorithm that was used for the MM in the original isoTPS [2]. The algorithm is sketched in figure 3.7 and is made up of three main steps.

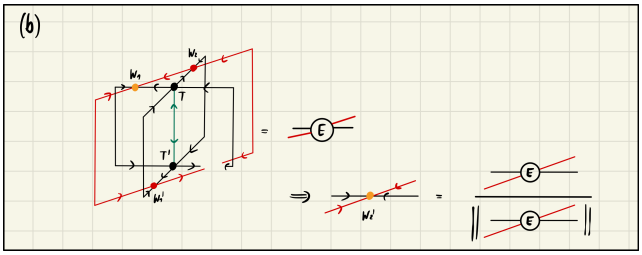
1. We start by contracting the tensors T, W_1 and W_2 into a single tensor Ψ. This tensor is then split from left to right via a truncated SVD

   $$\Psi = XSZ^\dagger = X \left(SZ^\dagger \right) =: X\theta$$

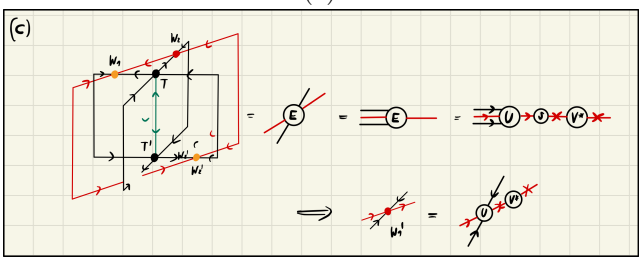
   as shown in figure 3.7(a). The bond dimension is truncated to D^2.



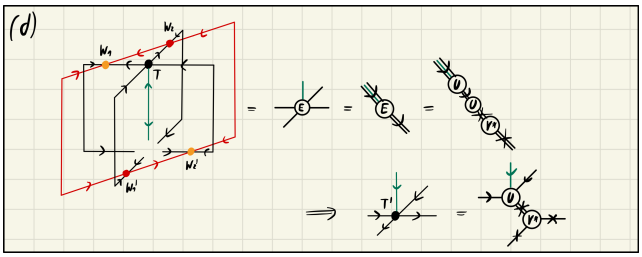
(a)



(b)



(c)



(d)

Figure 3.6: (a) The cost function of the optimization problem (3.3) can be computed as a contraction of only six tensors. (b) To optimize the tensor W_2', all tensors except W_2' are contracted into the environment E. The updated tensor is then given as $W_2' = E/\|E\|$. (c) The tensor W_1' can be updated by contracting all tensors except W_1' into the environment E, which is subsequently isometrized using an SVD. (d) The tensor T' can be updated similarly by contracting all tensors except T' into the environment E and isometrizing E using an SVD.

2. Next, we split the index of the bond connecting X and θ into two indices of dimension D each, see figure 3.7(b). To proceed, we note that there exists a degree of freedom on the bonds connecting X and θ: A unitary U and its adjoint can be inserted as shown in the second step of figure 3.7(b) without changing the result of the contraction

$$XU^\dagger U\theta = \left(XU^\dagger\right)(U\theta) =: T'\tilde{\theta}.$$

   This unitary U can be chosen to minimize the truncation error of the next step by *disentangling* the tensor θ. We will discuss procedures of finding such a *disentangling unitary* on the next page.

3. In the last step, the tensor $\tilde{\theta}$ is split vertically into W_1' and W_2' using a truncated SVD as shown in figure 3.7(c). Here, the bond dimension is truncated to χ. We end up with the three tensors T', W_1' and W_2', completing the YB move.

Before we discuss the disentangling procedure, two comments about step two of the above algorithm are in order. Firstly, there exists a degree of freedom for splitting the bond index, because applying the same permutations to the columns of X and rows of θ does not change the result of contracting the network. However, this degree of freedom is fixed by the disentangling process, making the exact permutation of the bond splitting irrelevant. Secondly, note that near the edges of the lattice it can happen that the matrizized tensor Ψ has $\tilde{\chi} < D^2$ rows. In this case, the bond dimension after the SVD will also be $\tilde{\chi}$ and we cannot simply split the bond into two bonds of dimension $\chi_1 = \chi_2 = D$. Instead, we choose a splitting $\chi_1 \leq D$, $\chi_2 \leq D$ such that $\chi_1 \cdot \chi_2$ is maximized, while it must still hold $\chi_1 \cdot \chi_2 \leq \tilde{\chi}$. We additionally prefer "equal" splittings $\chi_1 \approx \chi_2 \approx \sqrt{\tilde{\chi}}$ if possible. One can find such a splitting easily by computing all possible combinations of χ_1 and χ_2 and keeping only the best one. This has a computational cost of $\mathcal{O}\left(\sqrt{\tilde{\chi}}\right) = \mathcal{O}(D)$.

The disentangling process

We will now discuss the problem of finding a good disentangling unitary U for step 2 of the above algorithm, which is crucial for the performance of the YB move. The problem can be formulated as follows: Given the tensor θ that is obtained after splitting the index in step two, find a unitary U minimizing a cost function $f(U, \theta)$. In the following, let $\tilde{\theta}_{(l,i),(j,r)}$ be the $\chi D^2 \times \chi D^2$ matrix that is obtained by reshaping the contraction $\tilde{\theta}_{l,i,j,r} = \sum_{i',j'} U_{i,j,i',j'}\theta_{l,i,j,r}$ into a matrix as shown in figure 3.8. Let further $\tilde{\theta} = XSY$ denote the SVD of $\tilde{\theta}$. We discuss two cost

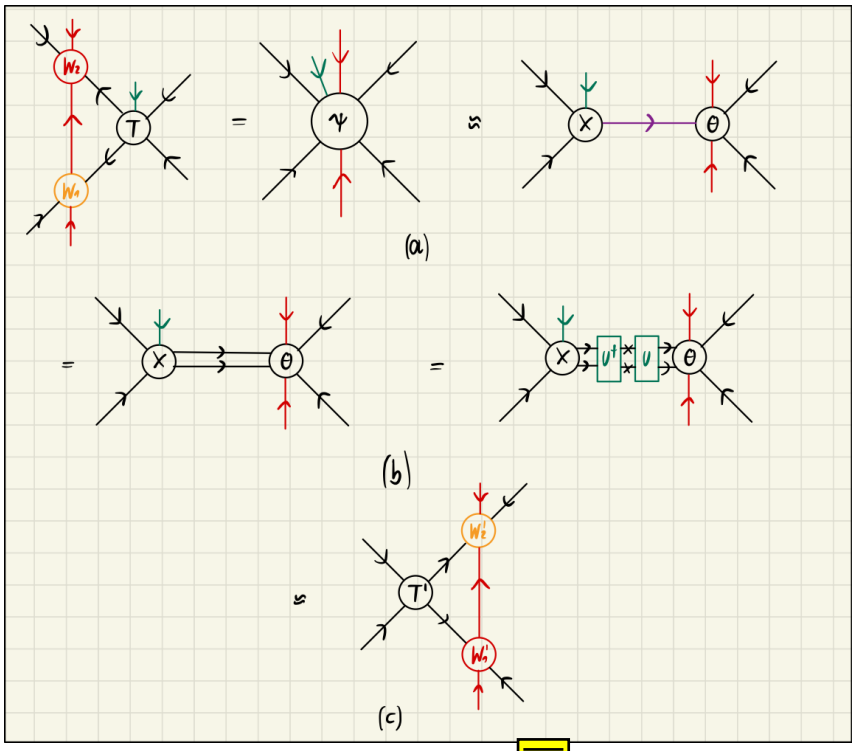


Figure 3.7: test

functions. The first cost function is simply given by the truncation error

$$f_{\text{trunc}}\left(U,\theta\right)=\sum_{j=\chi+1}^{\chi D^2}S_j^2 \tag{3.4}$$

arising in step three of the YB move. Alternatively, one can think of $|\tilde{\theta}\rangle \coloneqq \sum_{(l,i),(j,r)}\tilde{\theta}_{(l,i),(j,r)}|(l,i),(j,r)\rangle$ as a bipartite state in an orthogonal basis and consider as a cost function the Rényi-entropy

$$f_{\text{Rényi}}\left(U,\theta,\alpha\right)=\frac{1}{1-\alpha}\log\text{Tr}\left(\rho^{\alpha}\right)=\frac{1}{1-\alpha}\log\left(\sum_{j=1}^{\chi D^2}s_i^{2\alpha}\right), \tag{3.5}$$

where $\alpha\in[0,\infty)$ and $\rho=\text{Tr}_{(j,r)}\left(|\tilde{\theta}\rangle\langle\tilde{\theta}|\right)$ is the reduced density matrix obtained by tracing out one of the subsystems, see figure 3.9. In the last step of (3.5) we used the fact that the eigenvalues of ρ are the squares of the singular values of $\tilde{\theta}$. The Rényi-entropy can be used as a measure of entanglement. It approaches the

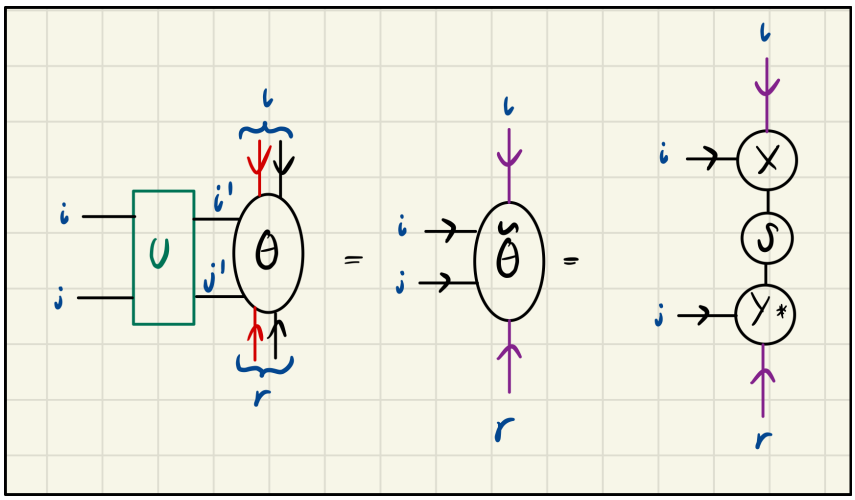


Figure 3.8: The disentangling unitary U is contracted with the wave function tensor θ to form $\tilde{\theta}$, which is subsequently split via an SVD $\tilde{\theta} = XSY^\dagger$.

Von-Neumann entanglement entropy for $\alpha \to 1$. It can be shown that the truncation error is bounded by the Rényi-entropy if $\alpha < 1$ [28], which is a motivation for using $f_{\text{Rényi}}$ as a cost function. For $\alpha > 1$ such a bond cannot generally be given. However, optimizations of Rényi-entropies with $\alpha > 1$ are often simpler to perform and still achieve good results in practice [1, 2, 29]. Setting $\alpha = 2$ yields the Rényi-entropy

$$f_{\text{Rényi}}(U, \theta, \alpha = 2) = -\log \operatorname{Tr} \rho^2,$$

which can easily computed by contracting the tensor network shown in figure 3.10(a) without needing to perform an SVD. The cost function $f_{\text{Rényi}}(U, \theta, \alpha = 2)$ can be minimized using the Evenbly-Vidal algorithm as proposed in [29]. First, the minimization problem can be rewritten as a maximization problem

$$U^{\text{opt}} = \underset{U^\dagger U = \mathbb{1}}{\operatorname{argmin}} f_{\text{Rényi}}(U, \theta, \alpha = 2) = \underset{U^\dagger U = \mathbb{1}}{\operatorname{argmax}} \operatorname{Tr} \rho^2.$$

We proceed by taking one tensor U out of the network $\operatorname{Tr} \rho^2$ and contracting all other tensors into the environment E as shown in figure 3.10(b). We now treat E as if it were independent of U and update $U \leftarrow AB^\dagger$, where A and B are obtained by taking the SVD $E = A\Lambda B^\dagger$. For details on the Evenbly-Vidal algorithm see appendix A.2. In practice it is observed that this algorithm for minimizing $f_{\text{Rényi}}(U, \theta, \alpha = 2)$ converges very quickly [2].

¸ Minimizing the truncation error $f_{\text{trunc}}(U, \theta)$ and general Rényi-entropies $f_{\text{Rényi}}(U, \theta, \alpha \neq 2)$ is a harder problem. We follow the approach of [1, 2] and use Riemannian optimization [30–33] to solve the optimization problem. The idea of

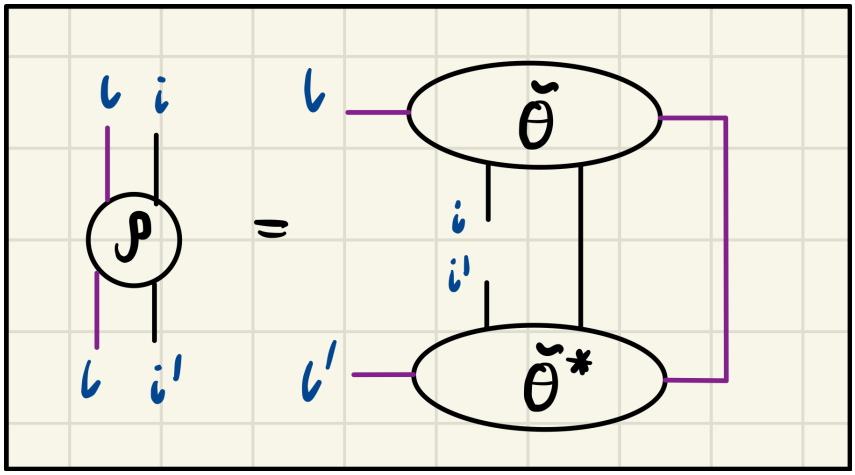


Figure 3.9: Definition of the reduced density matrix ρ.

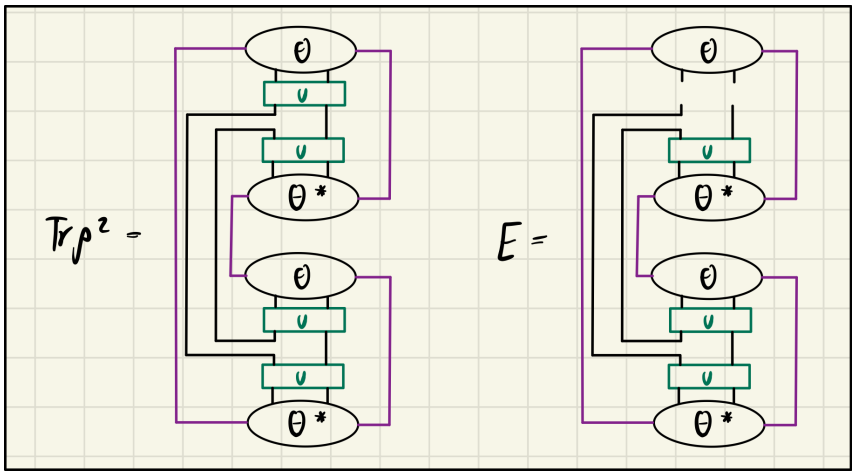


Figure 3.10: (a) Tensor network for the computation of the cost function (**??**). (b) Taking out one unitary U, the tensor network is contracted into the environment E.

Riemannian optimization is to generalize common optimization algorithms defined in Euclidian vector spaces, such as Gradient Descent or Conjugate Gradients, to Riemannian manifolds. The set of all isometric matrices of shape $n \times m$ is a Riemannian manifold called the Stiefel manifold $\mathrm{St}(n, p)$. A special case is the set of all unitary matrices of shape $n \times n$, $U(n) = \mathrm{St}(n, n)$, over which we want to optimize here.
A typical optimization algorithm iteratively improves an iterate $U_k \in \mathrm{St}(n, p), k = 1, 2, \ldots$ until a local minimum of the cost function f is found. The gradient of the cost function $\nabla f(U_k)$ is restricted to the tangent space $T_{U_k}\mathrm{St}(n, p)$ to the iterate U_k, which we visualize in figure 3.12. The gradient can be computed either analytically or via automatic differentiation [32, 33]. Optimization algorithms typically compute a search direction $\xi \in T_{U_k}\mathrm{St}(n, p)$ and a step size $\alpha \in \mathbb{R}$ from the gradient. In an optimization algorithm defined on an Euclidean vector space one would then move along this direction as

$$\tilde{U}_{k+1} = U_k + \alpha.$$

However, $\tilde{U}_{k+1}$ is in general not an element of the manifold. To ensure $U_{k+1} \in \mathrm{St}(n, p)$, one can introduce a *retraction* $R_\xi : \mathbb{R} \to \mathrm{St}(n, p)$. One can think of $R_\xi(\alpha)$ as moving along the direction of ξ while staying on the manifold. As α increases, we move further along the path defined by the retraction, with $R_\xi(0) = U_k$. Different retractions can be chosen, varying by how well they perform in optimization problems and by how hard they are to compute. Here we choose the retraction

$$R_\xi(\alpha) = \mathrm{qf}(U_k + \alpha\xi),$$

where $\mathrm{qf}(A)$ is the Q-factor of the QR-decomposition $A = QR$. This retraction is particularly easy to compute and yields good results in practice.
Many optimization algorithms such as Conjugate Gradients require gradients from previous iterates for computing a search direction at the current iterate. In Riemannian optimization, these gradients must first be brought from the tangent spaces of previous iterates to the tangent space of the current iterate. This is handled by a so-called *vector transport*, for more details see appendix B.
Finally, for optimization algorithms of second order such as the trust-region method, one needs to generalize the notion of the hessian-vector product to Riemannian manifolds. This generalization is given by the *Riemannian connection* [30]. For the Stiefel manifold this is simply given by projecting the hessian vector product of the embedding euclidean vector space $\mathbb{C}^{n \times p}$ to the tangent space, see appendix **??**.
We used two algorithms for solving the disentangling optimization problem, Conjugate Gradients (CG) and the Trust-Region Method (TRM). Conjugate gradients is a first order method which uses the accumulated gradients of previous iterations to compute a search direction, trying to achieve faster convergence than

simple gradient descent. CG is discussed in more details in appendix **??**. The TRM approximates the cost function around the current iterate through a quadratic function using the hessian vector product. This approximate cost function is then minimized within a region of radius $\Delta \in \mathbb{R}$ using truncated Conjugate Gradients (tCG), which converges quickly for the quadratic approximation. Depending on the quality of the approximation at the current iterate one can then shrink or enlarge the trust region. TRM is able to achieve superlinear convergence on many cost functions. For more details, see appendix **??**.
The gradients and hessian vector products of the cost functions (3.4) and (**??**) can be computed analytically and are drawn as tensor network diagrams in figure **??**. A derivation of these results is given in appendix **??**.

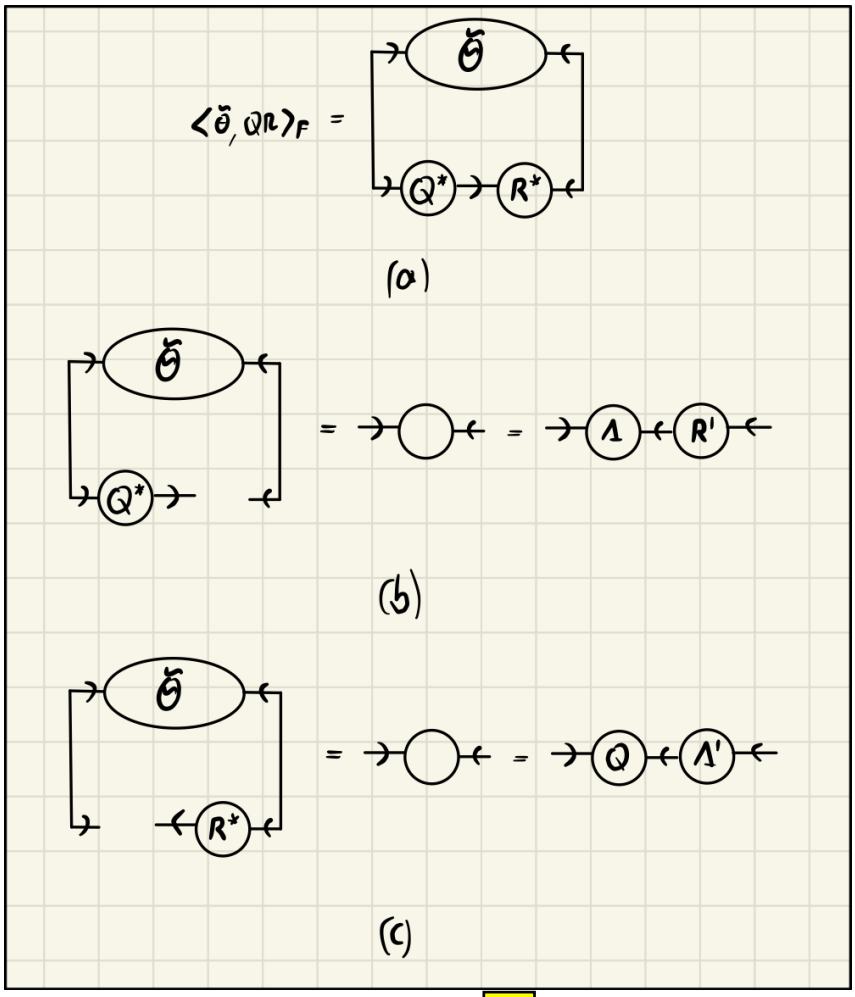
Approximate gradients and hessian vector products

The cost of both CG and the TRM are dominated by the computation of the gradient and hessian vector product, particularly by the SVD $\tilde{\theta} = XSY$ and contractions involving X and Y. The reason for this is the large bond dimension χD^2 of the bond connecting X and Y. We thus propose to approximate the gradient and hessian vector product by only performing an approximate SVD $\tilde{\theta} \approx \tilde{X}\tilde{S}\tilde{Y}$. The algorithm for performing this approximate SVD is inspired by [], where a similar algorithm was used to speed up TEBD updates for MPS. We sketch the algorithm in figure **??**. First, an approximate QR-decomposition $\tilde{\theta} = QR$ is performed by variationally minimizing the distance $\|\tilde{\theta} - QR\|$, alternatingly optimizing the tensors Q and R as shown in figures 3.11(b) and 3.11(c) respectively. A standard SVD is then performed on the R-factor of the approximate QR-decomposition as $R = A\tilde{S}\tilde{Y}$, and the contraction $\tilde{X} = QA$ finalizes the decomposition. It is observed that the variational minimization converges very quickly in practice, especially if a good initialization is chosen. As the iterates U_k are expected to only change slightly each iteration of CG or TRM, one can simply use the result Q, R obtained in the approximate QR-decomposition of the previous iteration as initialization for the current iteration. We observe that the variational optimization converges in ≤ 5 iterations in most cases.

Comparison of different disentangling algorithms

We will now compare the different algorithms for solving the disentangling problem.

For this comparison we select a YB move environment $\{W_1, W_2, T\}$ that was encountered during a ground state search of the transverse field Ising model using imaginary TEBD on disoTPS, see secion **??** for details. The results on other YB environments agree qualitatively with the results we present here. The bond dimensions chosen for the disoTPS are $D = 4$, $\chi = 24$.



centering

Figure 3.11:

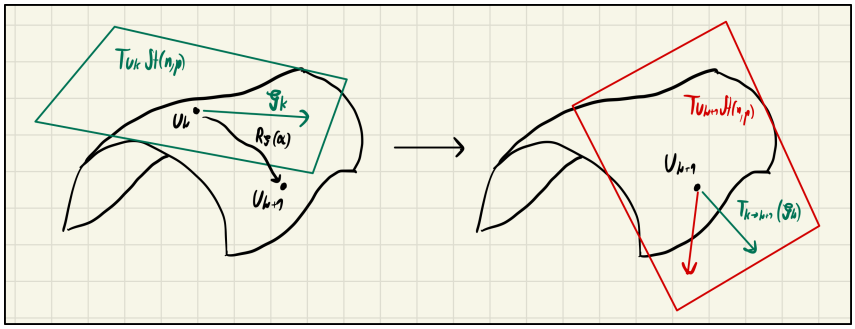


Figure 3.12: In this figure, a visualization of optimization on Riemannian manifolds is given. The iterate U_k (left) is updated along the search direction ξ_k, which is an element of the tangent space $T_{U_k}\mathrm{St}(n,p)$. The next iterate U_{k+1} is computed with the retraction $R_\xi(\alpha)$, where $\alpha \in \mathbb{R}$ is the step size. For the computation of the next search direction ξ_{k+1} the previous search direction ξ_k is needed, which is brought to the tangent space $T_{U_{k+1}}\mathrm{St}(n,p)$ of the new iterate U_{k+1} via the vector transport $T_{k\to k+1}(\xi_k)$.

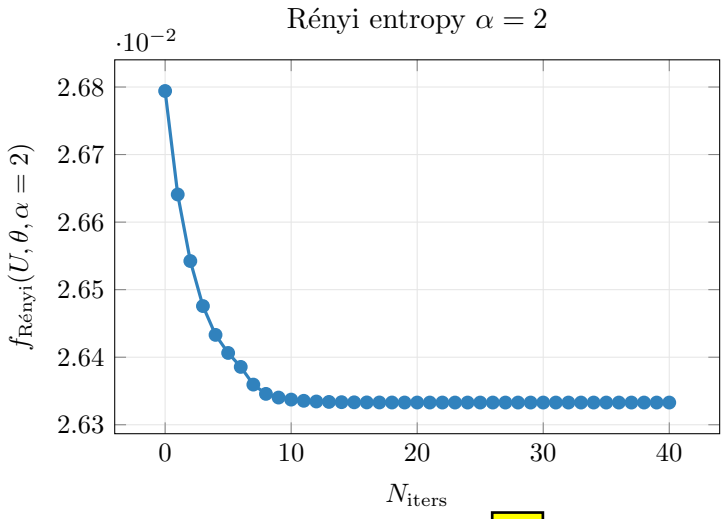


Figure 3.13:

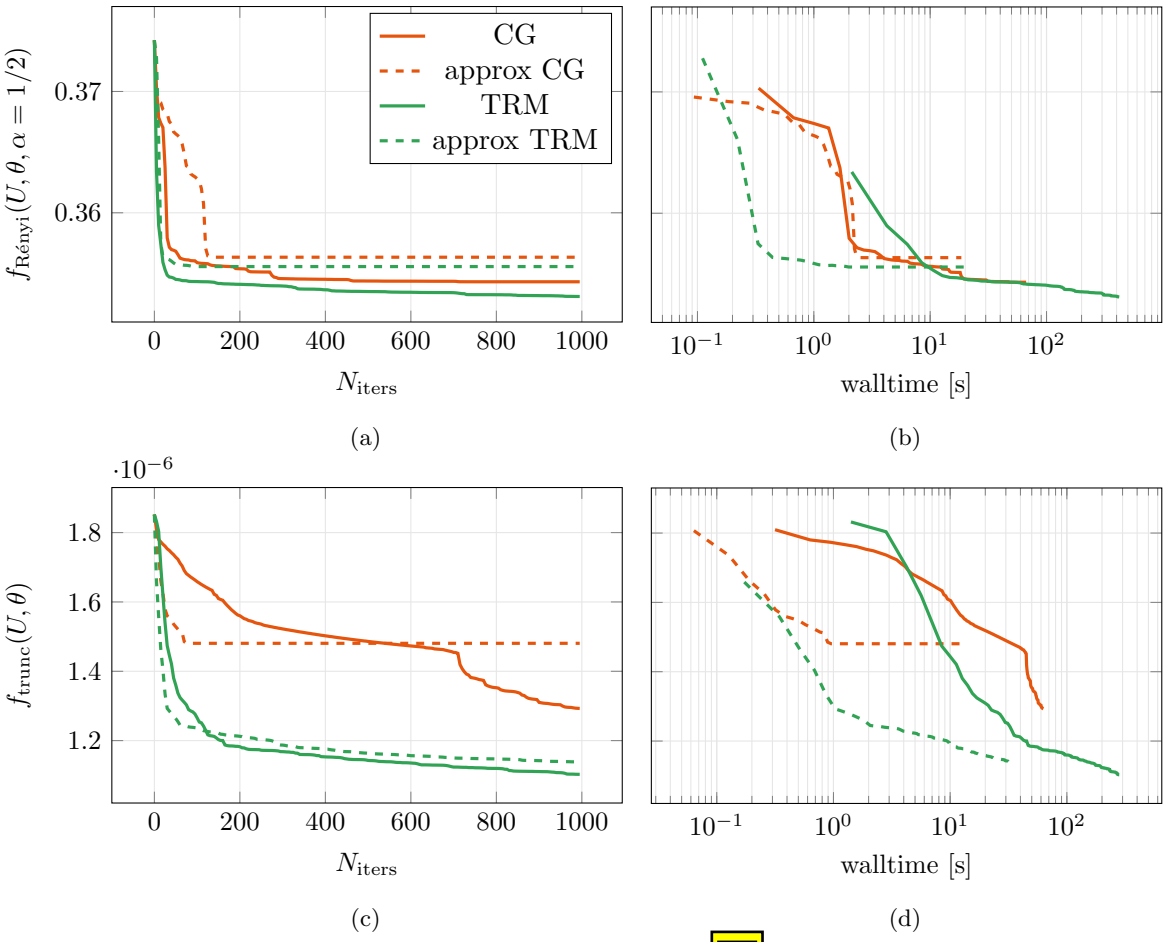


Figure 3.14:

We first test the Evenbly-Vidal algorithm optimizing the Rényi-entropy with $\alpha = 2$, see figure 3.13. The algorithm converges very quickly after only ≈ 20 iterations. The convergence speed depends drastically on the initialization of the disentangling unitary. We find that a initialization based on an SVD of θ works best, see appendix **??**.

Next we look at the performance of CG and TRM optimizing the Rényi-entropy with $\alpha = 1/2$ and the truncation error in figures 3.14a and 3.14c respectively. In both cases we observe that the TRM leads to a faster decrease in the cost function for the same number of iterations compared to CG. Remarkably, the approximate versions of both CG and TRM perform not drastically worse than their exact counterparts. Again, the speed of convergence depends strongly on the initialization. Because of the quick convergence of the Evenbly-Vidal algorithm optimizing the

$\alpha = 2$ Rényi-entropy we choose its result as an initialization for the optimization algorithms using Riemannian optimization. This achieved the best results in our testing.
Lastly we plot the cost function against the walltime of the algorithms in figures 3.14b and 3.14d. For this benchmark, the algorithms were run on a i5-12500 CPU with 6 cores. As one can see, the approximate versions of both CG and TRM run up to an order of magnitude faster while still providing a comparable minimization of the cost function. Thus, in practice, it is often better to choose a larger maximum bond dimension with the approximate disentangling algorithms instead of a smaller maximum bond dimension with the exact algorithms.

3.2.3 Comparison of the two algorithms

3.3 Time Evolving Block Decimation (TEBD)

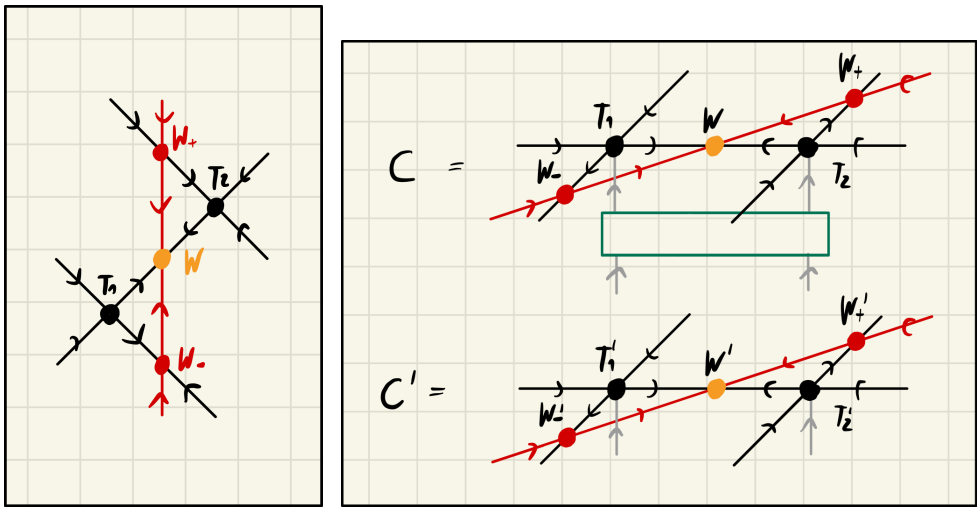
We will now discuss the Time Evolving Block Decimation (TEBD) algorithm for disoTPS, which can be used for both real- and imaginary time evolution. The algorithm is a generalization of TEBD for MPS, which we discussed in section **??**. Analogeously to MPS we start with a Suzuli-Trotter decomposition, approximating the time evolution operator $U(\Delta t) = e^{-i\Delta t\hat{H}}$ by a product of bond operators $U^{[x,y]}(\Delta t)$ acting only on neighbouring sites on the bond $[x, y]$. These bond operators then must be applied to the state in the correct order, while keeping the disoTPS structure intact. We will discuss the process of applying a single bond operator $U^{[x,y]}(\Delta t)$ to the disoTPS in section **??**. In section **??** we then discuss the full TEBD algorithm.

3.3.1 Local TEBD updates

Let us assume that the orthogonality center is positioned between the two sites on which the bond operator $\hat{U}^{[x,y]}(\Delta t)$ acts. The five tensors around the orthogonality center then make up a sub-network with only incoming arrows, see figure 3.15a. We call these five tensors T_1, T_2, W_1, W_2 and W_3. The local TEBD update can then be formulated as the following problem: Find tensors T_1', T_2', W_1', W_2' and W_3' satisfying the isometry constraints and minimizing the error

$$\varepsilon_{\text{trunc}} = \left\| \hat{U}^{[x,y]}(\Delta t)|\Psi\rangle - |\Psi\rangle \right\| = \underset{T_1',T_2',W_1',W_2',W_3'}{\operatorname{argmax}} \operatorname{Re}\langle \mathcal{C}|\mathcal{C}'\rangle_{\text{F}}. \tag{3.6}$$

The tensors $\mathcal{C}$ and $\mathcal{C}'$ denote the contracted sub-networks that are defined in figure 3.15b. For solving this problem we use the Evenbly-Vidal algorithm, inspired by [34], where a similar algorithm was used to speed up TEBD for MPS. As we already did in section 3.2.1 for the YB move, we optimize one tensor at a time while keeping all



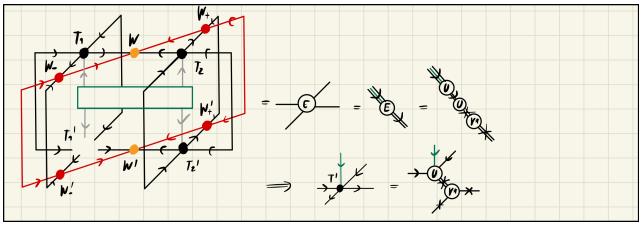
(a) (b)

Figure 3.15: (a The five tensor environment of the bond on which the bond operator is to be applied. All arrows are incoming. (b) Definitions of the sub-networks $\mathcal{C}$ and $\mathcal{C}'$ used in equation (3.6).

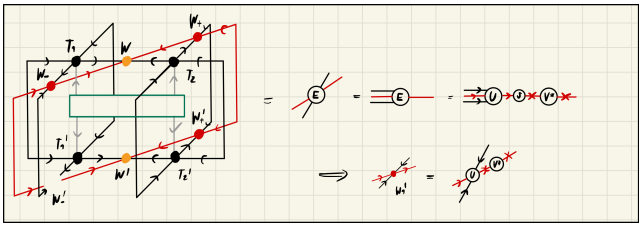
other tensors fixed. This procedure is then repeated, sweeping over all five tensors until convergence is achieved. For more details on this optimization method see appendix B. One iteration of the algorithm is shown in figure 3.16.

3.3.2 Global TEBD updates

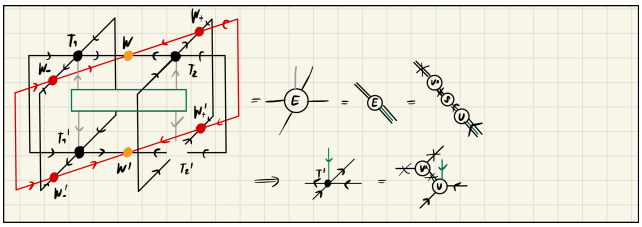
A global TEBD update evolves the state by a time Δt and can be performed by applying local TEBD updates on all bonds. For each local TEBD update, the orthogonality center must be moved to the bond at which the update is applied. Because moving the orthogonality hypersurface can only be done approximately, we would like to minimize the number of necessary moves.
As we have already done for MPS in section **??** let us assume that the Hamiltonian $\hat{H}$ can be written as a sum of nearest-neighbour operators. We now index these nearest-neighbour operators $h^{[x,y]}$ by two integers x and y corresponding to the position of the orthogonality hypersurface and orthogonality center if moved to the bond on which $h^{[x,y]}$ acts. We define x to increase from left to right and y to increase from bottom to top, as shown in figure 3.17a. The Hamiltonian can then be split into four parts by first grouping the $h^{[x,y]}$ into two sets acting only on even and odd columns respectively and then splitting each set again into terms acting only on



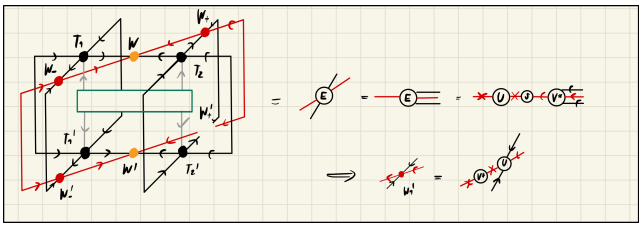
(a)



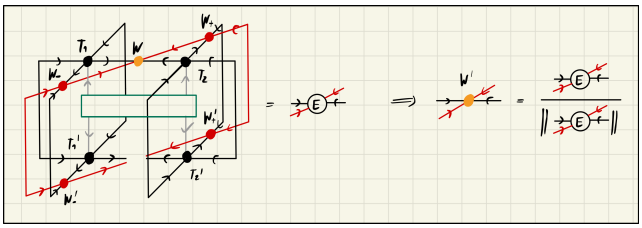
(b)



(c)

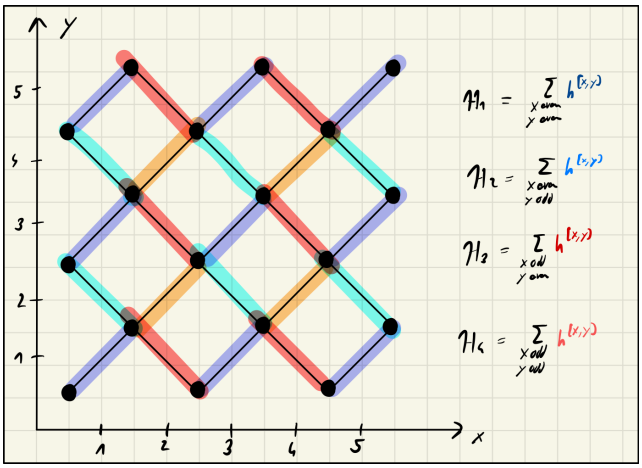


(d)

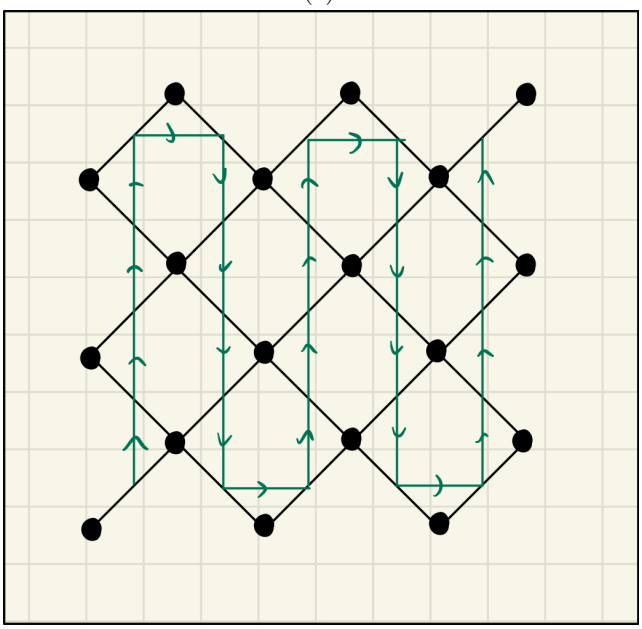


(e)

Figure 3.16: A local TEBD update can be applied by variational optimization of five tensors. Subfigures (a)-(e) show steps 1-5 of one iteration of the algorithm, updating each tensor once.



(a)



(b)

Figure 3.17: (a) A Hamiltonian $\hat{H}$ that is a sum of nearest-neighbour operators $h^{[x,y]}$ can be split into four parts made up of operators acting only on even/odd columns and even/odd bonds along a column. (b) A second order Suzuki-Trotter decomposition results in a chain of bond operators that have to be applied in the order visualized, moving along the path from the start to the end and backwards to the start again.

even/odd bonds along the respective columns. We can write this as

$$\hat{H} = \sum_{x=1}^{2L_x-1} \sum_{y=1}^{2L_y-1} h^{[x,y]} = \sum_{\substack{x \text{ even} \\ y \text{ even}}} h^{[x,y]} + \sum_{\substack{x \text{ even} \\ y \text{ odd}}} h^{[x,y]} + \sum_{\substack{x \text{ odd} \\ y \text{ even}}} h^{[x,y]} + \sum_{\substack{x \text{ odd} \\ y \text{ odd}}} h^{[x,y]}$$
$$=: \hat{H}_1 + \hat{H}_2 + \hat{H}_3 + \hat{H}_4,$$

see also figure 3.17a. The operators appearing in the sum in $\hat{H}_j$ commute with each other and thus the exponential $e^{-i\Delta t \hat{H}_j}$ factorizes into a product of bond operators $\hat{U}^{[x,y]}(\Delta t) = e^{-i\Delta t h^{[x,y]}}$.
Similar to section **??** we can use a Suzuki-Trotter decomposition to approximate the time evolution operator

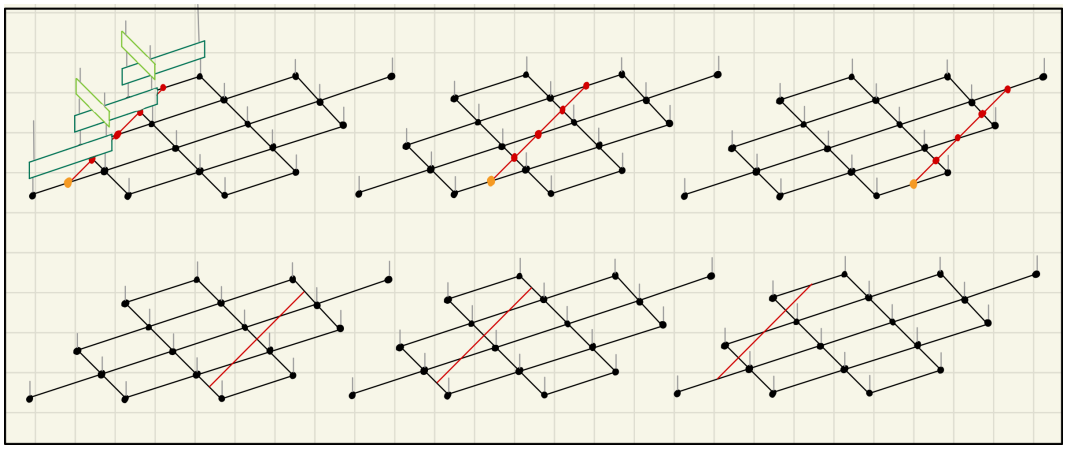
$$\hat{U}(\delta t) = \hat{U}^{\text{TEBD1}}(\Delta t) + \mathcal{O}(\Delta t^2)$$

with

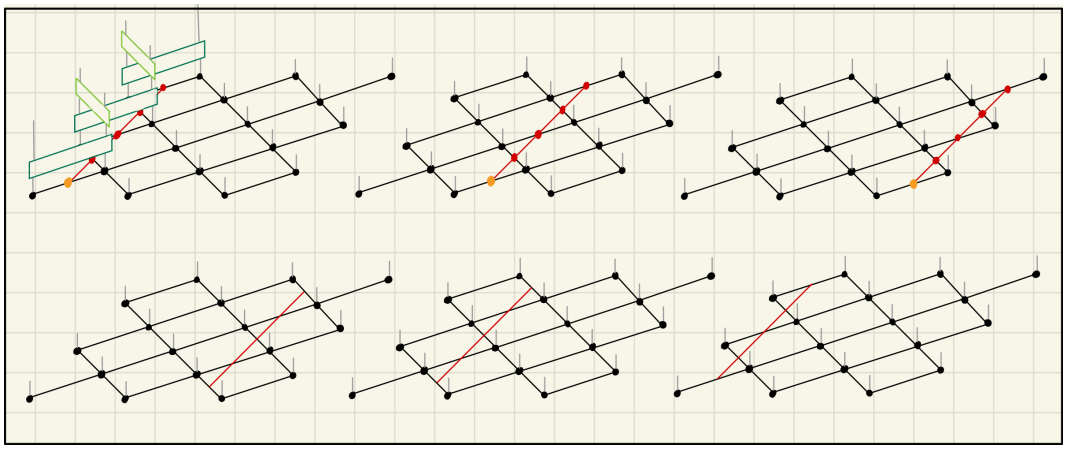
$$\hat{U}^{\text{TEBD}}(\Delta t) \coloneqq e^{-i\Delta t \hat{H}_4} e^{-i\Delta t \hat{H}_3} e^{-i\Delta t \hat{H}_2} e^{-i\Delta t \hat{H}_1}.$$

To evolve the state $|\Psi\rangle$ in time with this first order approximation we must compute $|\Psi'\rangle \approx U^{\text{TEBD1}}(\Delta t)|\Psi\rangle$ as a disoTPS. The procedure is sketched in figure 3.18a. We start in the left-most column and apply all bond operators that act on bonds along this column. The bond operators on the column are applied analogeously to the MPS algorithm: First we update all even bonds and then we update all odd bonds. Local updates are computed using the algorithm discussed in section 3.3.1. Next, we move the orthogonality two columns to the right and again apply all bond operators along the column. We proceed until all bond operators on even columns have been applied, in which case the orthogonality hypersurface is now positioned at its right-most position. We now sweep back to the left, applying all bond operators acting on odd columns along the way. Arriving back at the left-most column, all bond making up $\hat{U}^{\text{TEBD1}}(\Delta t)$ have been applied in the correct order and the state has been evolved by time Δt. We can obtain a better approximation of the time evolution operator $U(\Delta t)$ by performing a second order Suzuki-Trotter decomposition. By repeatedly applying the symmetrized decomposition

$$e^{-i\varepsilon(A+B)} = e^{-i\frac{\varepsilon}{2}A} e^{-i\frac{\varepsilon}{2}A} e^{-i\varepsilon B} + \mathcal{O}(\Delta t^3)$$



(a)



(b)

Figure 3.18: To apply a TEBD update of (b) first order and (b) second order, we sweep accross the disoTPS once from left to right and back from right to left. (a) TEBD1 applies the operators in a brick-wall fashion, while (b) TEBD2 applies the operators along a chain.

we obtain

$$\begin{aligned} e^{-i\Delta t\hat{H}} &= \exp\left(-i\Delta t\sum_{x,y}\hat{h}^{[x,y]}\right) \\ &= e^{-i\frac{\Delta t}{2}\hat{h}^{[1,1]}}e^{-i\Delta t\left(\hat{H}-\hat{h}^{[1,1]}\right)}e^{-i\frac{\Delta t}{2}\hat{h}^{[1,1]}} + \mathcal{O}(\Delta t^3) \\ &= e^{-i\frac{\Delta t}{2}\hat{h}^{[1,1]}}e^{-i\frac{\Delta t}{2}\hat{h}^{[1,2]}}e^{-i\Delta t\left(\hat{H}-\hat{h}^{[1,1]}-\hat{h}^{[1,2]}\right)}e^{-i\frac{\Delta t}{2}\hat{h}^{[1,2]}}e^{-i\frac{\Delta t}{2}\hat{h}^{[1,1]}} + \mathcal{O}(\Delta t^3) \\ &= \cdots \\ &= e^{-i\frac{\Delta t}{2}\hat{h}^{[1,1]}}e^{-i\frac{\Delta t}{2}\hat{h}^{[1,2]}}\cdots e^{-i\frac{\Delta t}{2}\hat{h}^{[1,2]}}e^{-i\frac{\Delta t}{2}\hat{h}^{[1,1]}} + \mathcal{O}(\Delta t^3). \end{aligned}$$

Here, in each step we "split off" one operator $\hat{h}^{[x,y]}$ from the sum. The final result is a product of bond operators $\hat{U}^{[x,y]}(\Delta t/2))$ that must be applied from right to left. We can visualize this as a chain of bond operators that are applied along the path sketched in figure 3.17b. We start from the bottom left of the lattice, visiting every bond, and ending up on the top right. The same string of bond operators must then be applied backwards until arriving again at the bottom left.
The algorithm of applying a global update is thus similar to a TEBD update of first order. We sweep across the disoTPS once from left to right and back, applying bond operators along the way in the correct order, as visualized in figure 3.18b. The number of YB moves for TEBD1 and TEBD2 is the same, but the smaller Trotter error of $\mathcal{O}(\Delta t^3)$ instead of $\mathcal{O}(\Delta t^2)$ allows us to use larger time steps for TEBD2, resulting in a smaller number of YB moves per unit time. We find that the YB move is the primary source of error in practice and thus expect TEBD2 to perform much better than TEBD1. In principle, one could also go to higher decomposition orders [13]. However, already a third order decomposition would necessitate a larger number of sweeps for applying the full update, increasing the error accumulated through YB moves. It is therefore not clear if higher order decompositions would be able to improve the method further.

Chapter 4

Toric Code: An exactly representable Model

4.1 The Toric Code Model

The Toric Code is an exactly soluble spin model with $\mathbb{Z}_2$ topological order that was introduced by Alexei Kitaev [35]. The model is defined on the square lattice with periodic boundary conditions, where on each edge of the lattice there sits a spin-1/2 degree of freedom. Two operators are introduced, the *star operators*

$$A_+ \coloneqq \sum_{j \in +} \hat{\sigma}_j^z \tag{4.1}$$

and the *plaquette operators*

$$A_\square \coloneqq \sum_{j \in \square} \hat{\sigma}_j^x, \tag{4.2}$$

where the sums are performed over the four spins connected in a star or plaquette pattern respectively (see figure 4.1) and $\hat{\sigma}_j^x, \hat{\sigma}_j^z$ are Pauli matrices. The Hamiltonian of the Toric Code model is then defined as

$$H_{\mathrm{TC}} \coloneqq -\sum_{+} A_+ - \sum_{\square} B_\square, \tag{4.3}$$

where the sums go over all possible stars and plaquettes respectively. Because an arbitrary star and plaquette operator share either two or zero spins, all terms of the Hamiltonian commute and it is thus possible to find the ground state of the model by diagonalizing all terms simultaneously. To diagonalize the star operators A_+ we choose a basis of $\hat{\sigma}_z$-eigenstates $|i\rangle \in \{|\uparrow\rangle, |\downarrow\rangle\}$ with eigenvalues $\langle \hat{\sigma}_z \rangle_i = s_i = \pm 1$ for each spin i. In this basis every star operator is diagonal with eigenvalues

$$\langle A_+ \rangle = \prod_{j \in +} s_j \in \{1, -1\}.$$

To obtain the expectation value $\langle A_+ \rangle = 1$, the number of spins in the down state $s_j = -1$ around the vertex $+$ must be even. Basis states $|s\rangle$ that give an expectation

value of 1 for every star operator simultaneously are thus the states with an even number of down-spins around every vertex,

$$|s\rangle = |s_1\rangle \otimes \cdots \otimes |s_N\rangle\,, \quad \prod_{j\in+} s_j = 1 \;\; \forall + \,. \tag{4.4}$$

The plaquette operator $B_\square$ acts on a basis state by flipping all spins around the plaquette $\square$. Because a plaquette and a star share either zero or two spins, applying an plaquette operator to a state $|s\rangle$ satisfying condition (4.4) produces a state $|s'\rangle$ that again satisfies (4.4), and applying the plaquette operator a second time produces the initial state $|s\rangle$. If we now take the equal weighted superposition $|\Psi\rangle = (|s\rangle + |s'\rangle)/\sqrt{2}$, the expectation value of $B_\square$ becomes $\langle B_\square\rangle = 1$. The ground state of the Hamiltonian (4.3) is thus given by the equal weighted superposition of all basis states satisfying condition (4.4).
One can show that the ground state can be written as

$$|\Psi_0\rangle \propto \prod_\square (\mathbb{1} + B_\square)\,|\uparrow\rangle \otimes \cdots \otimes |\uparrow\rangle\,.$$

Note that this is also the ground state for the model if open boundary conditions are chosen instead.
For periodic boundary conditions, which is equivalent to putting the model on a torus, one can further show that the ground state is fourfold topologically degenerate. To move from one degenerate section of the Hilbert space to another a string of operators must be applied, wrapping once around the torus. This is a highly non-local operation. Because perturbarions are usually local, the toric code model can be interpreted as a form of "hardware level" error correction. The toric code is considered a topological quantum error correction code and can in theory be used for quantum "memory". One can further implement quantum gates acting on the 4-dimensional ground state space by locally creating a pair of anyonic excitations, moving one of the excitations around the torus, and annihilating it with the other one [35]. Unfortunately, the gates that can be implemented as such do not form a complete state set and thus do not allow for universal quantum computing. Nevertheless, the Toric code is an important model for the study of topological order and anyonic excitations.

4.2 Representing the Toric Code Ground State with disoTPS

We will now derive the disoTPS corresponding to the Toric Code ground state on a square lattice with open boundary condition. We choose rough boundary condition [36], fixing all boundary spins to the state $(|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$. As shown in [19], the Toric Code ground state can be represented exactly as a PEPS with bond dimension

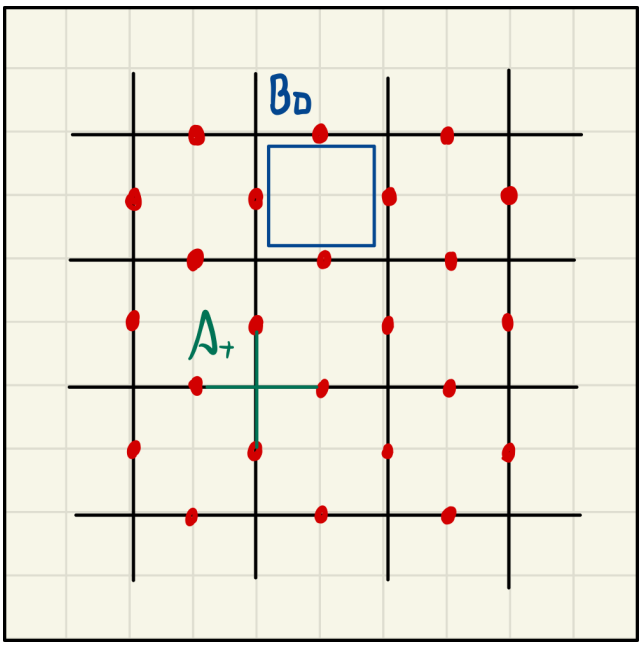
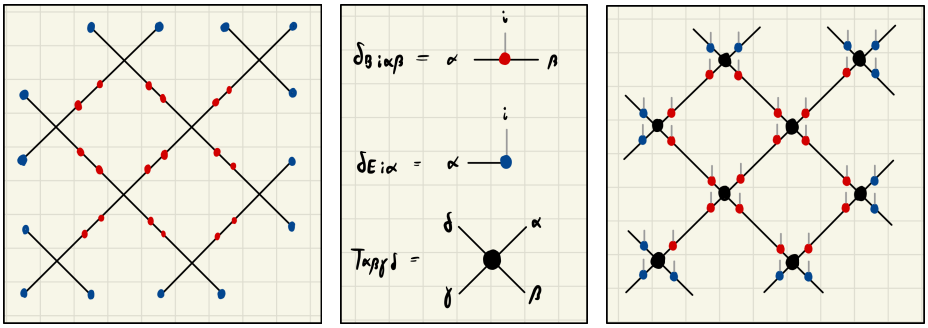
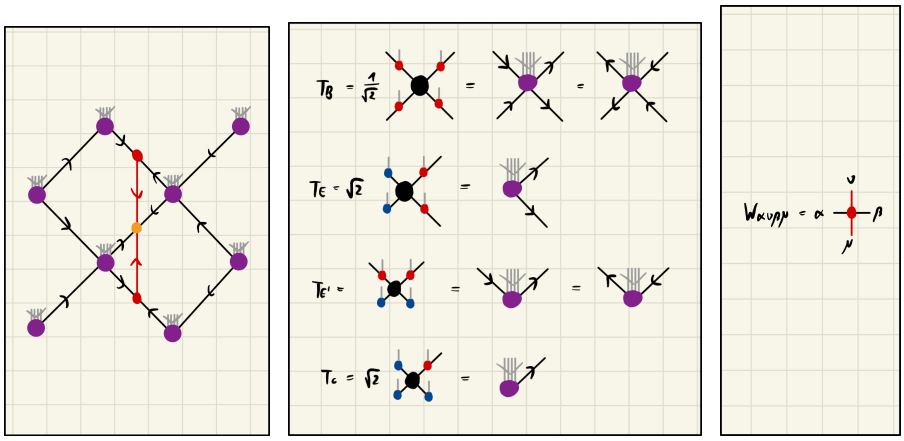


Figure 4.1: The Toric Code model is defined on the square lattice with spin-1/2 degrees of freedom living on the edges. the star and plaquette operators (4.1) and (4.2) act on the four spins aranged in a star or plaquette shape respectively.



(a) (b) (c)

Figure 4.2: (a) To represent the Toric Code ground state as a PEPS we start by doubling the local degrees of freedom on each edge in the bulk. Bulk spins are denoted in red, while boundary spins are colored blue. (b) Tensor diagrams of the tensors δ^{B}, δ^{E} and T introduced in the text. (b) The PEPS representation of the Toric Code ground state before contracting the tensors at each vertex, made up from the tensors (b).



(a) (b) (c)

Figure 4.3: The PEPS in figure 4.2c can be transformed to the disoTPS (a) by normalizing the tensors as shown in (b). Note that the tensors T'_{E} at the top and bottom edges of the lattice need a different normalization than the tensors T_{E} at the left and right edges. The tensors of the orthogonality surface are defined in (c).

$D = 2$. One can construct such a PEPS easily by first doubling the Hilbert space on each edge as $|s_i\rangle \to |s_i\rangle \otimes |s_i\rangle$, which we show in figure 4.2a. In the PEPS representation the physical degrees of freedom on each edge in the bulk are then carried by two identical tensors $\delta^{\mathrm{B}} \in \mathbb{R}^{2\times2\times2}$,

$$\delta^{\mathrm{B}} i, \alpha, \beta = \begin{cases} 1 & \text{if } i = \alpha = \beta \\ 0 & \text{else} \end{cases},$$

as shown in figure 4.2b. The boundary spins are represented by tensors $\delta_E \in \mathbb{R}^{2\times2}$,

$$\delta^{\mathrm{E}} i, \alpha = \begin{cases} 1 & \text{if } i = \alpha \\ 0 & \text{else} \end{cases}.$$

We proceed by associating each vertex with the spins on the four connected edges and connect the corresponding tensors δ^{B} and δ^{E} with a tensor $T \in \mathbb{R}^{2\times2\times2\times2}$ that is placed on each vertex,

$$T_{i,j,k,l} = \begin{cases} 1 & \text{if } (i+j+k+l) \mod 2 = 0 \\ 0 & \text{else} \end{cases}.$$

This tensor ensures that all states with an odd number of down spins around a vertex have an amplitude of zero, satisfying condition (4.4).

We arrive at the PEPS in figure 4.3c. Each basis state satisfying condition (4.4) results in the same amplitude when contracting the PEPS, while basis states violating the condition vanish. Thus, the PEPS represents the ground state of the Toric Code.
We now want to transform the PEPS into a disoTPS. This can be easily done by choosing the correct normalization for the vertex tensors, which transforms them into isometries as shown in figure 4.3b. Note that different normalizations need to be choosen for tensors at the corners, edges, and in the bulk. For each vertex tensor we can choose the isometry direction to point to the left or to the right respectively, allowing us to place the orthogonality hypersurface anywhere in the lattice. As a last step, the tensors of the orthogonality hypersurface must be specified. The two spins that are connected to a tensor W of the orthogonality hypersurface must be in the same local state, since they were created by doubling the local degree of freedom. This constraint can be enforced by setting $W \in \mathbb{R}^{2\times1\times2\times1}$ to

$$W_{\alpha,\nu,\beta,\mu} = \frac{\delta_{\alpha,\beta}}{\sqrt{2}}$$

with dummy indices ν, μ of bond dimension 1. Trivially the tensors W are isometries as shown in figure 4.3c. We can again choose the direction of isometry to point either up or down for every W-tensor, allowing us to place the orthogonality center freely along the orthogonality hypersurface.
We have thus found an exact disoTPS representation of the Toric Code ground state with $D = 2$ and $\chi = 1$, similar to the construction done in [19] for isoTPS. The final network is depicted in figure 4.3a. We test the different algorithms for the YB move on the Toric Code ground state on a 5×5 lattice. All algorithms are able to move the orthogonality surface exactly up to computational accuracy. This will however only work if a good initialization is chosen for the optimization algorith  We discuss this initialization for the disentangling unitary in appendix **??**

Chapter 5

Transverse Field Ising Model: Ground State Search and Time Evolution

5.1 The Transverse Field Ising Model

5.2 Ground State Search

5.3 Time Evolution after a Global Quench

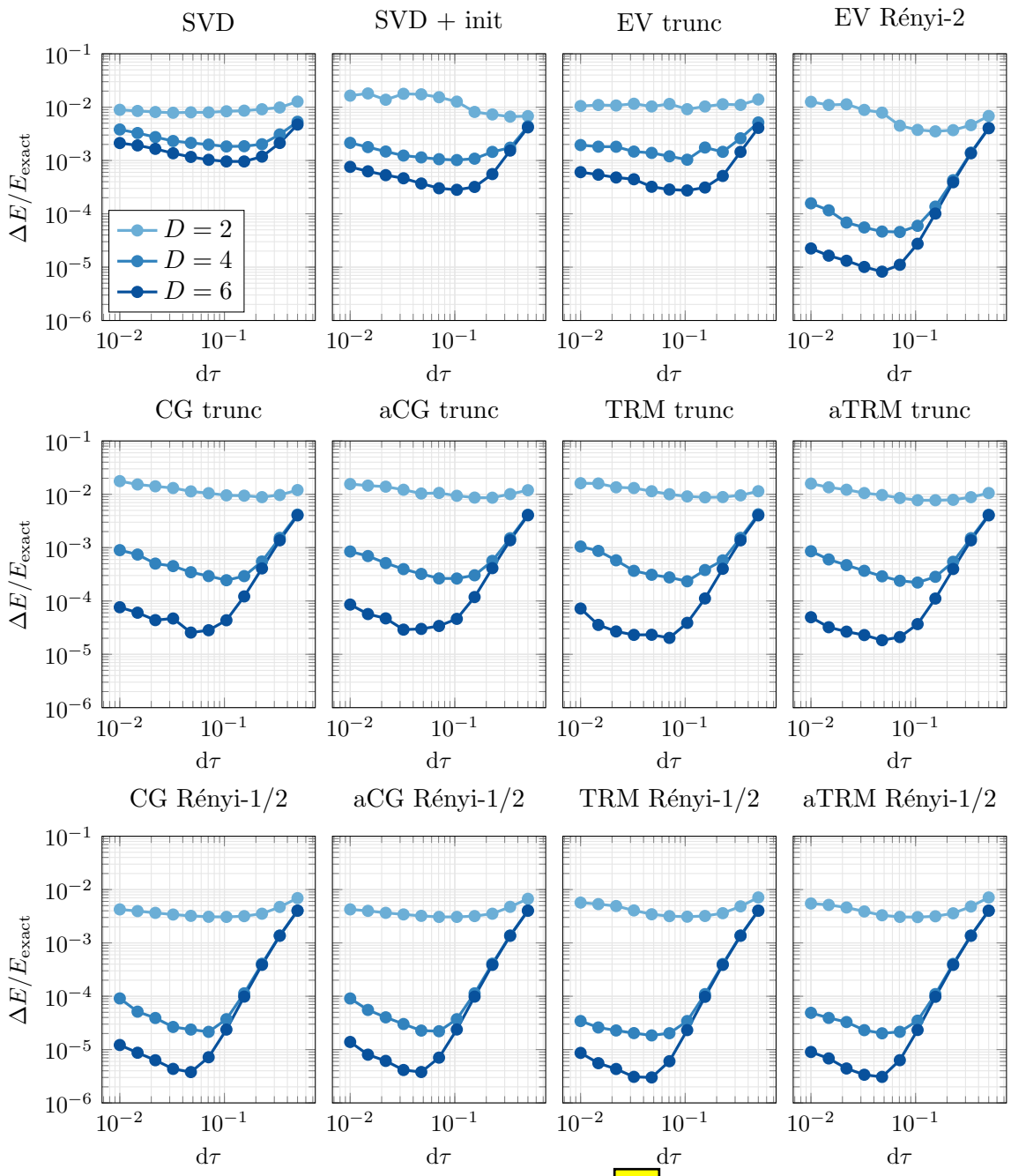

SVD
SVD + init
EV trunc
EV Rényi-2
CG trunc
aCG trunc
TRM trunc
aTRM trunc
CG Rényi-1/2
aCG Rényi-1/2
TRM Rényi-1/2
aTRM Rényi-1/2
$\Delta E/E_{\text{exact}}$
$D = 2$
$D = 4$
$D = 6$
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-2}
10^{-1}
$d\tau$

Figure 5.1:

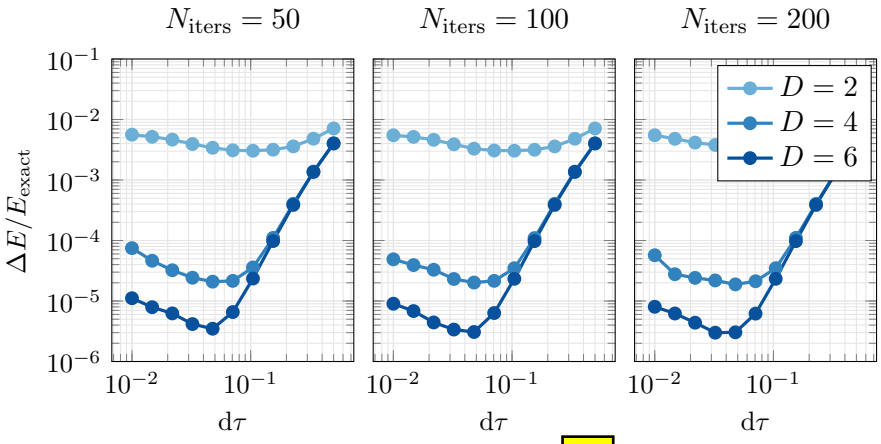


Figure 5.2:

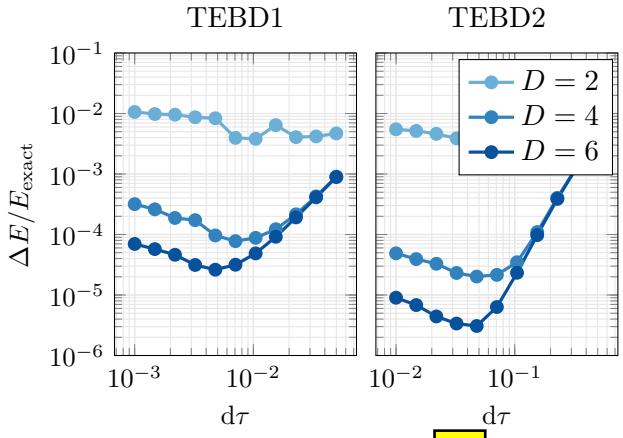


Figure 5.3:

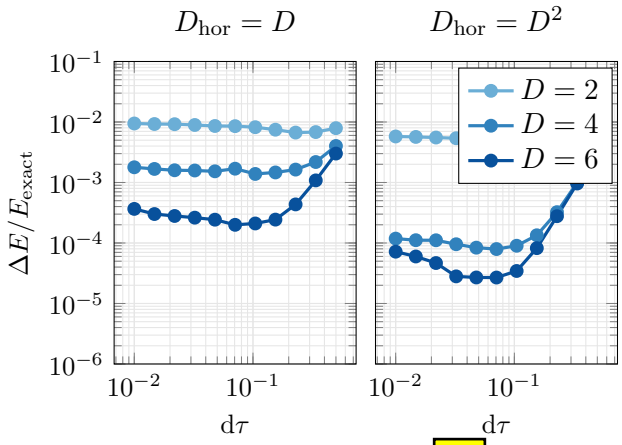


Figure 5.4:

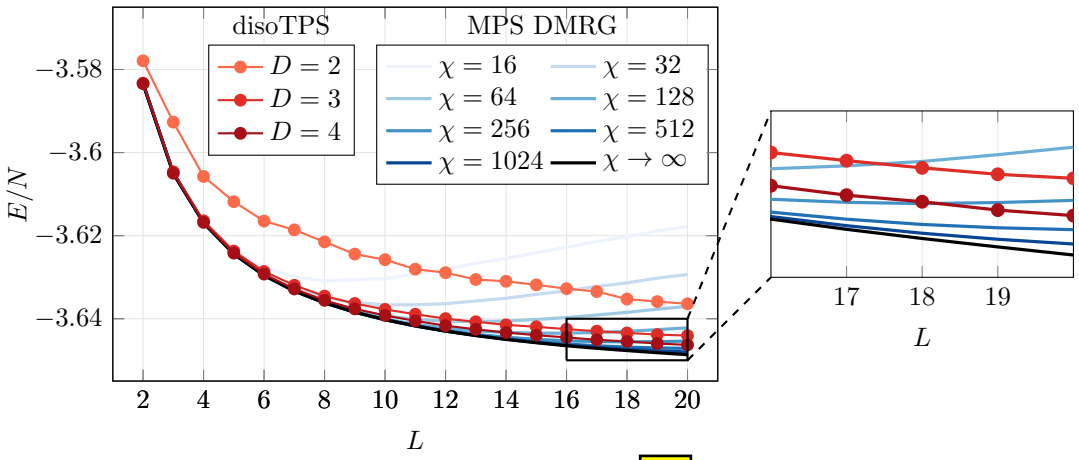


Figure 5.5:

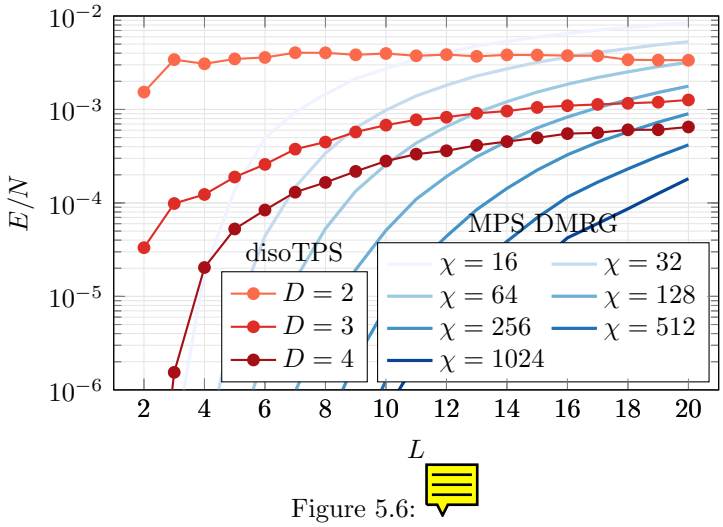

10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
E/N
2
4
6
8
10
12
14
16
18
20
L
disoTPS
$D = 2$
$D = 3$
$D = 4$
MPS DMRG
$\chi = 16$
$\chi = 32$
$\chi = 64$
$\chi = 128$
$\chi = 256$
$\chi = 512$
$\chi = 1024$

Figure 5.6:

Appendix A

Optimization Problems for isometric Tensor Networks

When discussing algorithms on isometric tensor networks, one often needs to find optimal tensors extremizing a given cost function f. In the most general case, f is a function

$$f : \mathbb{C}^{m_1 \times n_1} \times \cdots \times \mathbb{C}^{m_K \times n_K} \to \mathbb{R},$$

mapping K tensors $T_1, \ldots, T_K$ to a scalar cost value. Here, the tensors have already been reshaped into matrices, grouping together legs with incoming arrows and legs with outgoing arrows respectively. The tensors must satisfy certain constraints. If a tensor T_i posesses both legs with incoming arrows and legs with outgoing arrows, it must satisfy the isometry constraint $T_i^\dagger T_i = \mathbb{1}$, where without loss of generalization we assumed $n_i \geq m_i$. If instead the tensor T_j posesses only legs with incoming arrows (and thus is an orthogonality center), it is constrained to be normalized to one, $\|T_j\|_\mathrm{F} = 1$. To summarize, we want to solve the optimization problem

$$T_1^\mathrm{opt}, \ldots, T_K^\mathrm{opt} = \underset{T_1, \ldots T_K}{\mathrm{argmax}} f\left(T_1, \ldots T_K\right) \tag{A.1}$$

under the constraints

$$T_i^\dagger T_i = \mathbb{1}$$

for isometries T_i and

$$\|T_j\|_\mathrm{F} = 1$$

for the orthogonality center T_j.
In the following, we will discuss several approaches for solving optimization problem (A.1). We will first assume that the input of the cost function is a single tensor T. If the cost function is linear, the problem is known as the *orthogonal Procrustes problem* and we discuss its closed form solution in section A.1. Non-linear cost functions can be optimized by using the Evenbly-Vidal algorithm, see section A.2. Finally, we will discuss how cost functions of multiple tensors can be optimized in section A.3.
A different, more involved approach to solving the optimization problem is given by Riemannian optimization, which we discuss in appendix B.

A.1 The orthogonal Procrustes problem

If the cost function is linear, it can be written as

$$f(T) = \sum_{i=1}^{m}\sum_{j=1}^{n} \left[\alpha_{i,j} \operatorname{Re}\left(T_{i,j}\right) + \beta_{i,j} \operatorname{Im}\left(T_{i,j}\right)\right]$$

with parameters $\alpha_{i,j}, \beta_{i,j} \in \mathbb{R}$. Introducing the *environment tensor* $E \in \mathbb{C}^{m\times n}$ as $E_{i,j} = \alpha_i + \imath\beta_j$ we can write the cost function as

$$f(T) = \sum_{i=1}^{m}\sum_{j=1}^{n} \operatorname{Re}\left(E^*_{i,j}T_{i,j}\right) = \operatorname{Re}\operatorname{Tr}\left(E^\dagger T\right) = \operatorname{Re}\operatorname{Tr}\left(T^\dagger E\right).$$

Maximizing $f(T)$ under the isometry constraint $T^\dagger T = \mathbb{1}$ is known as the orthogonal Procrustes problem and permits the closed form solution

$$T^{\text{opt}} = \underset{T^\dagger T=\mathbb{1}}{\operatorname{argmax}} \operatorname{Re}\operatorname{Tr}\left(T^\dagger E\right) = UV^\dagger, \tag{A.2}$$

where the matrices U and V are computed using an SVD $E = USV^\dagger$. To prove this result we insert the SVD into the cost function as

$$\begin{aligned} f(T) &= \operatorname{Re}\operatorname{Tr}\left(ET^\dagger\right) = \operatorname{Re}\operatorname{Tr}\left(USV^\dagger T^\dagger\right) = \operatorname{Re}\operatorname{Tr}\left[\left(U\sqrt{S}\right)\left(\sqrt{S}V^\dagger T^\dagger\right)\right] \\ &= \operatorname{Re}\left\langle \sqrt{S}U^\dagger, \sqrt{S}V^\dagger T^\dagger \right\rangle_{\text{F}}. \end{aligned}$$

We next use the fact that the Frobenius inner product satisfies the Cauchy-Schwarz inequality to obtain the upper bound

$$\begin{aligned} f(T) &= \operatorname{Re}\left\langle \sqrt{S}U^\dagger, \sqrt{S}V^\dagger T^\dagger \right\rangle_{\text{F}} \leq \left\| \sqrt{S}U^\dagger \right\|_{\text{F}} \left\| \sqrt{S}V^\dagger T^\dagger \right\|_{\text{F}} \\ &= \sqrt{\operatorname{Tr}\left(USU^\dagger\right)\operatorname{Tr}\left(TVSV^\dagger T^\dagger\right)} = \operatorname{Tr}\left(S\right), \end{aligned}$$

where in the last step we used $U^\dagger U = \mathbb{1}$, $V^\dagger V = \mathbb{1}$, $T^\dagger T = \mathbb{1}$ and the cyclic property of the trace. This upper bound is reached by the solution

$$F\left(T^{\text{opt}}\right) = \operatorname{Re}\operatorname{Tr}\left(USV^\dagger VU^\dagger\right) = \operatorname{Tr}\left(S\right),$$

proving (A.2).
If the tensor T must not satisfy the isometry condition but must be normalized to one, the closed form solution can be found as

$$T^{\text{opt}} = \underset{\|T\|=1}{\operatorname{argmax}} \operatorname{Re}\operatorname{Tr}\left(T^\dagger E\right) = E/\left\|E\right\|. \tag{A.3}$$

We arrive at this solution through a aimilar argument as before. First, we obtain an upper bound

$$f(T) = \mathrm{Re}\,\mathrm{Tr}\left(T^\dagger E\right) = \mathrm{Re}\,\langle T, E\rangle_{\mathrm{F}} \leq \|T\|\,\|E\| = \|E\|$$

using the Cauchy-Schwarz inequality and the normalization constraint $\|T\| = 1$. We proceed by showing that the upper bound is reached by T^{opt},

$$f(T^{\mathrm{opt}}) = \mathrm{Re}\,\mathrm{Tr}\left(EE^\dagger / \|E\|\right) = \|E\|\,,$$

proving (A.3).

A.2 The Evenbly-Vidal algorithm

In general the cost function $f(T)$ is not linear. For example, a non-linear cost function is encountered in the disentangling procedure when optimizing a MERA wave function []. It was proposed by Evenbly and Vidal [] to linearize the cost function and to update the tensor T iteratively using the closed form solutions from section A.1. Let us assume that the cost function $f(T)$ can be written as the contraction of a tensor network, where in general the tensor T may appear multiple times. We contract all tensors except one of the tensors T into an environment tensor $E_T \in \mathbb{C}^{n\times n}$ and the cost function becomes

$$f(T) = \mathrm{Re}\,\mathrm{Tr}\,(E_T T)\,.$$

We now keep the environment E_T fixed, trating it as if it were independant of T, and updating T with the closed form solutions (A.2) or (A.3). This is repeated until T is converged. If and how fast T converges depends on the details of the cost fucntion, but connvergence cannot be guaranteed for arbitrary cost functions. This procedure is discussed in more detail and for general cost functions in appendix **??** One can also use Riemannian optimization for the optimization of general non-linear cost functions of isometries. This method is more powerful but also more involved and is discussed in appendix B.

A.3 Cost functions of multiple tensors

cost functions of multiple tensors $T_1, \ldots T_K$ can be optimized iteratively via an algorithm similar to the Evenbly-Vidal algorithm. The idea is to optimize one tensor atr a time, keeping all other tensors fixed. To optimize the tensor T_i, we again contract all other tensors into an environment tensor E. If the environment tensor is independent of T_i (i.e. if the cost function is linear in T_i), we can update the tensor

with the closed form solutions of section A.1. Such an update is locally optimal in the sense that it maximizes $f(T_1, \ldots, T_K)$ for fixed tensors T_j, $j \neq i$. If the environment tensor is dependant of T_i, we need to use the Evenbly-Vidal algorithm (see section A.2) or Riemannian optimization (see appendix B). If the cost function is linear in all tensors $T_1, \ldots, T_K$ and bounded $f(T_1, \ldots, T_K) \leq c \in \mathbb{R}$, this algorithm is guaranteed to converge, since each local update is optimal and thus the cost function can never decrease.
An alternative approach for optimizing a cost function of multiple tensors is given by Riemannian optimization over product manifolds [31], which we discuss briefly in appendix **??**

Appendix B

Riemannian Optimization of Isometries

In this appendix we provide a brief introduction to the problem of optimizing a cost function on the constrained set of isometric matrices. This problem can be solved by performing Riemannian Optimization on the matrix manifold of isometric matrices, which is called the Stiefel manifold. For a more in-depth introduction to the topic we recommend the excellent book [30]. A discussion of Riemannian optimization of complex matrix manifolds in the context of quantum physics and isometric tensor networks can be found at [31, 32]. An implementation of Riemannian Optimization on the real Stiefel manifold and other matrix manifolds in python is given in [33]. Some parts of this implementation were also used in our implementation.

B.1 The complex Stiefel manifold

We define the *complex Stiefel manifold* $\mathrm{St}(n,p)$ with $n \geq p$ as the set of all isometric $n \times p$ matrices:

$$\mathrm{St}(n,p) \coloneqq \left\{ X \in \mathbb{C}^{n\times p} : X^\dagger X = \mathbb{1} \right\}.$$

In particular, for $n = p$, the complex Stiefel manifold reduces to the set of unitary matrices $U(n)$. One can show, similar to [30], that the complex Stiefel manifold is naturally an embedded submanifold of the Euclidian vector space $\mathbb{C}^{n\times p} \cong \mathbb{R}^{2np}$ of general complex $n \times p$ matrices.
Tangent vectors on manifolds generalize the notion of directional derivatives. A mathematical definition of tangent vectors and tangent spaces of manifolds is given in [30]. The set of all tangent vectors to a point $X \in \mathrm{St}(n,p)$ is called the *tangent space* $T_X\mathrm{St}(n,p)$, which is given by [30, 31]

$$T_X\mathrm{St}(n,p) = \left\{ Z \in \mathbb{C}^{n\times p} : X^\dagger Z + Z^\dagger X = 0 \right\}.$$

An arbitrary element $\xi \in \mathbb{C}^{n\times p}$ from the embedding space $\mathbb{C}^{n\times p}$ can be projected to the tangent space $T_X\mathrm{St}(n,p)$ by [30, 31]

$$P_X\xi = \xi - \frac{1}{2} X \left(X^\dagger \xi + \xi^\dagger X \right). \tag{B.1}$$

Additionally, we will also need to define a notion of length that we can apply to tangent vectors. This can be done in the form of an *inner product* on tangent spaces, called the *Riemannian metric*. A natural metric for the tangent space $T_X\mathrm{St}(n,p)$ of the Stiefel manifold is the Euclidean metric of the embedding space $\mathbb{C}^{n\times p}$, which is given by the real part of the Frobenius inner product:

$$g_W : T_X\mathrm{St}(n,p) \times T_X\mathrm{St}(n,p) \to \mathbb{R}, \quad g_X(\xi_1,\xi_2) = \mathrm{Re}\,\mathrm{Tr}\left(\xi_1^\dagger \xi_2\right). \tag{B.2}$$

Equipped with a Riemannian metric the Stiefel manifold becomes a Riemannian submanifold of $\mathbb{C}^{n\times p}$.
With these definition, we can now formulate the optimization problem as the problem of finding the isometry $W_{\mathrm{opt}} \in \mathrm{St}(n,p)$ that minimizes the cost function

$$f : \mathrm{St}(n,p) \to \mathbb{R}, \quad X \mapsto f(X). \tag{B.3}$$

B.2 Gradients, retractions, and vector transport

First order optimization algorihms like Gradient Descent and Conjugate Gradients use the gradient of the cost function to update the search direction at each iteration. In the case of the Stiefel manifold and the cost function (B.3), we first define the matrix of partial derivatives $D \in \mathbb{C}^{n\times p}$ of f at $X \in \mathrm{St}(n,p)$ by

$$D_{ij} \coloneqq \left.\frac{\partial f}{\partial \,\mathrm{Re}\,(X_{ij})}\right|_X + \mathrm{i}\left.\frac{\partial f}{\partial \,\mathrm{Im}\,(X_{ij})}\right|_X. \tag{B.4}$$

With this definition, the directional derivative $\mathrm{D}f(X)[Z]$ at $X \in \mathrm{St}(n,p)$ in direction $Z \in \mathbb{C}^{n\times p}$ is simply given by an inner product of D with the direction Z, using the Riemannian metric (B.2):

$$\begin{aligned} g_X(D,Z) &= \mathrm{Re}\,\mathrm{Tr}\left(D^\dagger Z\right) = \mathrm{Re}\sum_{ij} D_{ij}^* Z_{ij} \\ &= \sum_{ij}\left(\left.\frac{\partial f}{\partial \,\mathrm{Re}\,(X_{ij})}\right|_X \mathrm{Re}\,Z_{ij} + \left.\frac{\partial f}{\partial \,\mathrm{Im}\,(X_{ij})}\right|_X \mathrm{Im}\,Z_{ij}\right) \\ &\eqqcolon \mathrm{D}f(X)[Z]. \end{aligned}$$

With this we can now define the gradient $\nabla f(X)$ of f at $X \in \mathrm{St}(n,p)$ as the projection of the partial derivative matrix (B.4) to the tangent space [30, 31]:

$$\nabla f(X) \coloneqq P_X D = D - \frac{1}{2} X\left(X^\dagger D + D^\dagger X\right),$$

where we used the projection (B.1).

B.3 Conjugate Gradients

B.4 Trust Region Method

Appendix C

Initialization of the Disentangling Unitary

Appendix D

Computing the Gradient and Hessian of the Disentangling Cost Functions

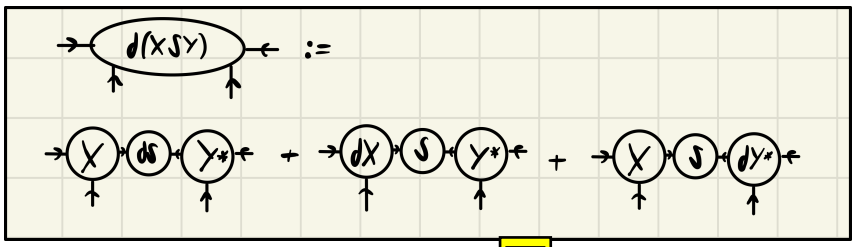


Figure D.1:

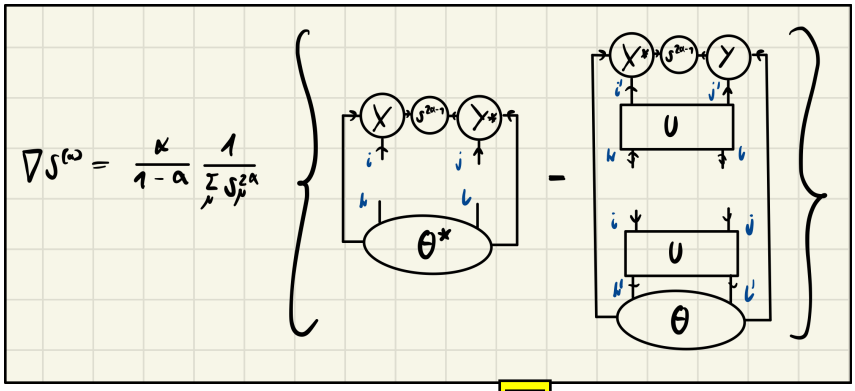


Figure D.2:

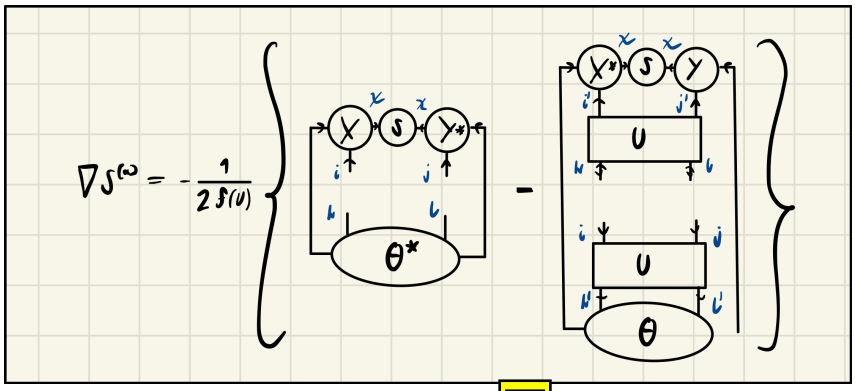


Figure D.3:

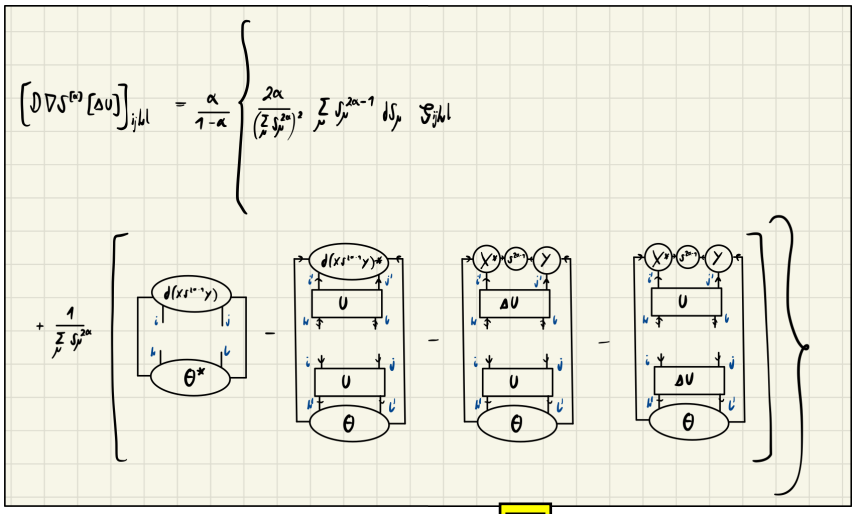


Figure D.4:

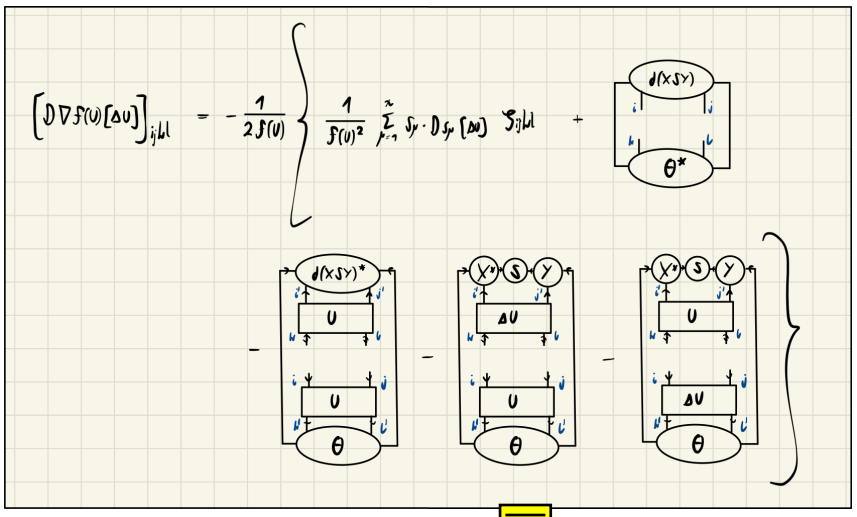


Figure D.5:

Bibliography

[1] Michael P. Zaletel and Frank Pollmann. 'Isometric Tensor Network States in Two Dimensions'. In: *Phys. Rev. Lett.* 124 (3 Jan. 2020), p. 037201. DOI: 10.1103/PhysRevLett.124.037201. URL: https://link.aps.org/doi/10.1103/PhysRevLett.124.037201.

[2] Sheng-Hsuan Lin, Michael P. Zaletel and Frank Pollmann. 'Efficient simulation of dynamics in two-dimensional quantum spin systems with isometric tensor networks'. In: *Phys. Rev. B* 106 (24 Dec. 2022), p. 245102. DOI: 10.1103/PhysRevB.106.245102. URL: https://link.aps.org/doi/10.1103/PhysRevB.106.245102.

[3] Carl Eckart and Gale Young. 'The approximation of one matrix by another of lower rank'. In: *Psychometrika* 1.3 (1936), pp. 211–218.

[4] Ulrich Schollwöck. 'The density-matrix renormalization group in the age of matrix product states'. In: *Annals of physics* 326.1 (2011), pp. 96–192.

[5] Román Orús. 'A practical introduction to tensor networks: Matrix product states and projected entangled pair states'. In: *Annals of physics* 349 (2014), pp. 117–158.

[6] Johannes Hauschild and Frank Pollmann. 'Efficient numerical simulations with Tensor Networks: Tensor Network Python (TeNPy)'. In: *SciPost Phys. Lect. Notes* (2018), p. 5. DOI: 10.21468/SciPostPhysLectNotes.5. URL: https://scipost.org/10.21468/SciPostPhysLectNotes.5.

[7] M B Hastings. 'An area law for one-dimensional quantum systems'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2007.08 (Aug. 2007), P08024. DOI: 10.1088/1742-5468/2007/08/P08024. URL: https://dx.doi.org/10.1088/1742-5468/2007/08/P08024.

[8] J. Eisert, M. Cramer and M. B. Plenio. 'Colloquium: Area laws for the entanglement entropy'. In: *Rev. Mod. Phys.* 82 (1 Feb. 2010), pp. 277–306. DOI: 10.1103/RevModPhys.82.277. URL: https://link.aps.org/doi/10.1103/RevModPhys.82.277.

[9] Guifré Vidal. 'Efficient simulation of one-dimensional quantum many-body systems'. In: *Physical review letters* 93.4 (2004), p. 040502.

[10] Frank Verstraete, Juan J Garcia-Ripoll and Juan Ignacio Cirac. 'Matrix product density operators: Simulation of finite-temperature and dissipative systems'. In: *Physical review letters* 93.20 (2004), p. 207204.

[11] Jutho Haegeman et al. 'Time-dependent variational principle for quantum lattices'. In: *Physical review letters* 107.7 (2011), p. 070601.

[12] Jutho Haegeman et al. 'Unifying time evolution and optimization with matrix product states'. In: *Physical Review B* 94.16 (2016), p. 165116.

[13] Naomichi Hatano and Masuo Suzuki. 'Finding exponential product formulas of higher orders'. In: *Quantum annealing and other optimization methods*. Springer, 2005, pp. 37–68.

[14] F. Verstraete et al. 'Criticality, the Area Law, and the Computational Power of Projected Entangled Pair States'. In: *Phys. Rev. Lett.* 96 (22 June 2006), p. 220601. DOI: 10.1103/PhysRevLett.96.220601. URL: https://link.aps.org/doi/10.1103/PhysRevLett.96.220601.

[15] RJ Baxter and IG Enting. 'Series expansions from corner transfer matrices: the square lattice ising model'. In: *Journal of Statistical Physics* 21 (1979), pp. 103–123.

[16] Michael Lubasch, J Ignacio Cirac and Mari-Carmen Banuls. 'Unifying projected entangled pair state contractions'. In: *New Journal of Physics* 16.3 (2014), p. 033014.

[17] Reza Haghshenas, Matthew J. O'Rourke and Garnet Kin-Lic Chan. 'Conversion of projected entangled pair states into a canonical form'. In: *Phys. Rev. B* 100 (5 Aug. 2019), p. 054404. DOI: 10.1103/PhysRevB.100.054404. URL: https://link.aps.org/doi/10.1103/PhysRevB.100.054404.

[18] Katharine Hyatt and E. M. Stoudenmire. *DMRG Approach to Optimizing Two-Dimensional Tensor Networks*. 2020. arXiv: 1908.08833 [cond-mat.str-el]. URL: https://arxiv.org/abs/1908.08833.

[19] Tomohiro Soejima et al. 'Isometric tensor network representation of string-net liquids'. In: *Phys. Rev. B* 101 (8 Feb. 2020), p. 085117. DOI: 10.1103/PhysRevB.101.085117. URL: https://link.aps.org/doi/10.1103/PhysRevB.101.085117.

[20] Daniel Malz and Rahul Trivedi. *Computational complexity of isometric tensor network states*. 2024. arXiv: 2402.07975 [quant-ph]. URL: https://arxiv.org/abs/2402.07975.

[21] Zhi-Yuan Wei, Daniel Malz and J. Ignacio Cirac. 'Sequential Generation of Projected Entangled-Pair States'. In: *Phys. Rev. Lett.* 128 (1 Jan. 2022), p. 010607. DOI: 10.1103/PhysRevLett.128.010607. URL: https://link.aps.org/doi/10.1103/PhysRevLett.128.010607.

[22] Lucas Slattery and Bryan K. Clark. *Quantum Circuits For Two-Dimensional Isometric Tensor Networks*. 2021. arXiv: 2108.02792 [quant-ph]. URL: https://arxiv.org/abs/2108.02792.

[23] Yu-Jie Liu, Kirill Shtengel and Frank Pollmann. *Topological quantum phase transitions in 2D isometric tensor networks*. 2024. arXiv: 2312.05079 [quant-ph]. URL: https://arxiv.org/abs/2312.05079.

[24] Zhehao Dai et al. *Fermionic Isometric Tensor Network States in Two Dimensions*. 2024. arXiv: 2211.00043 [cond-mat.str-el]. URL: https://arxiv.org/abs/2211.00043.

[25] Yantao Wu et al. 'Two-dimensional isometric tensor networks on an infinite strip'. In: *Phys. Rev. B* 107 (24 June 2023), p. 245118. DOI: 10.1103/PhysRevB.107.245118. URL: https://link.aps.org/doi/10.1103/PhysRevB.107.245118.

[26] Maurits S. J. Tepaske and David J. Luitz. 'Three-dimensional isometric tensor networks'. In: *Phys. Rev. Res.* 3 (2 June 2021), p. 023236. DOI: 10.1103/PhysRevResearch.3.023236. URL: https://link.aps.org/doi/10.1103/PhysRevResearch.3.023236.

[27] Wilhelm Kadow, Frank Pollmann and Michael Knap. 'Isometric tensor network representations of two-dimensional thermal states'. In: *Phys. Rev. B* 107 (20 May 2023), p. 205106. DOI: 10.1103/PhysRevB.107.205106. URL: https://link.aps.org/doi/10.1103/PhysRevB.107.205106.

[28] Frank Verstraete and J Ignacio Cirac. 'Matrix product states represent ground states faithfully'. In: *Physical Review B—Condensed Matter and Materials Physics* 73.9 (2006), p. 094423.

[29] Johannes Hauschild et al. 'Finding purifications with minimal entanglement'. In: *Phys. Rev. B* 98 (23 Dec. 2018), p. 235163. DOI: 10.1103/PhysRevB.98.235163. URL: https://link.aps.org/doi/10.1103/PhysRevB.98.235163.

[30] P.-A. Absil, R. Mahony and Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton: Princeton University Press, 2008. ISBN: 9781400830244. DOI: doi:10.1515/9781400830244. URL: https://doi.org/10.1515/9781400830244.

[31] Markus Hauru, Maarten Van Damme and Jutho Haegeman. 'Riemannian optimization of isometric tensor networks'. In: *SciPost Physics* 10.2 (Feb. 2021). ISSN: 2542-4653. DOI: 10.21468/scipostphys.10.2.040. URL: http://dx.doi.org/10.21468/SciPostPhys.10.2.040.

[32] Ilia A Luchnikov, Mikhail E Krechetov and Sergey N Filippov. 'Riemannian geometry and automatic differentiation for optimization problems of quantum physics and quantum technologies'. In: *New Journal of Physics* 23.7 (July 2021), p. 073006. ISSN: 1367-2630. DOI: 10.1088/1367-2630/ac0b02. URL: http://dx.doi.org/10.1088/1367-2630/ac0b02.

[33] James Townsend, Niklas Koep and Sebastian Weichwald. *Pymanopt: A Python Toolbox for Optimization on Manifolds using Differentiation*. 2016. arXiv: 1603.03236 [cs.MS].

[34] Jakob Unfried, Johannes Hauschild and Frank Pollmann. 'Fast time evolution of matrix product states using the QR decomposition'. In: *Physical Review B* 107.15 (2023), p. 155133.

[35] A Yu Kitaev. 'Fault-tolerant quantum computation by anyons'. In: *Annals of physics* 303.1 (2003), pp. 2–30.

[36] Alexei Kitaev and Liang Kong. 'Models for gapped boundaries and domain walls'. In: *Communications in Mathematical Physics* 313.2 (2012), pp. 351–373.