UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master degree in Physics of Data

Final dissertation

Tripartite entanglement decompositions for tensor networks

Thesis supervisor

Prof. Pietro Silvi

Thesis co-supervisor

Prof. Simone Montangero

Candidate

Filippo Pra Floriani

Academic year 2023/2024

Contents

Introduction

Understanding quantum many-body mechanics is a central challenge in modern physics, advancing both fundamental science and technology. Research in this field aims to discover and understand new phases of quantum matter, which are crucial for developing quantum technologies, quantum computing, and advanced information systems. These discoveries not only enhance our knowledge of high-energy physics but also have practical applications in areas such as chemistry, drug research, and the creation of new materials with unique properties.

Despite its significance, progresses in this area are constrained by the inherent complexity of many-body quantum systems. The scarcity of exactly solvable models and the limitations of current approximate approaches pose substantial obstacles. Overcoming these challenges is essential for advancing our comprehension of the fundamental building blocks of reality and for unlocking the potential of future technologies.

Over the last three decades, a powerful mathematical tool has emerged for enhancing our understanding of quantum many-body problems: Tensor Networks (TNs). Tensor networks provide an efficient framework for representing and manipulating high-dimensional tensors, which are essential for modeling complex quantum systems. The development of this framework has been instrumental in advancing quantum simulations, quantum algorithms, and various aspects of quantum information theory.

The origins of tensor networks can be traced back to the development of Matrix Product States (MPS) and Matrix Product Operators (MPO). These concepts arised from efforts to efficiently model one-dimensional quantum systems. In the early 1990s, the development of the Density Matrix Renormalization Group (DMRG) [1] algorithm marked a significant breakthrough. DMRG used a matrix product representation of quantum states to compute ground states and excited states of quantum systems with remarkable precision. This technique provided a way to manage the computational complexity associated with large quantum systems, laying the groundwork for more advanced tensor network methods.

The formalization of tensor networks gained momentum in the 2000s. During this period, researchers expanded upon the MPS concept, leading to the development of more general tensor network frameworks. These new frameworks allowed for the efficient representation of higher-dimensional quantum systems ([2]). The exploration of entanglement entropy and the discovery of the area law further advanced the theoretical understanding of tensor networks, contributing to their widespread adoption in quantum research ([3]).

One of the key development was the introduction of the Multiscale Entanglement Renormalization Ansatz (MERA) [4]. MERA is a tensor network ansatz designed to describe scale-invariant quantum states, providing insights into critical phenomena and quantum phase transitions. This method used a hierarchical structure of tensors to capture long-range correlations in quantum systems, which was crucial for studying quantum criticality.

In recent years, tensor networks have been applied to more complex and higher-dimensional quantum systems. Frameworks such as Projected Entangled Pair States (PEPS) [5] have been developed to address multidimensional systems and exotic quantum phases. Among the various tensor network structures, PEPS are particularly notable for their ability to efficiently represent the ground states

of local Hamiltonians in two or more dimensions. The PEPS framework is well-suited to capture the entanglement properties of quantum states, which are crucial for accurate simulations of quantum systems. Additionally, they naturally encode the area law of entanglement entropy, a key feature of ground states of local Hamiltonians in low-dimensional systems.

Despite the strengths of tensor networks, particularly PEPS, one significant challenge arises when dealing with loopy geometries. Loopy tensor networks, which include loops or cycles within their graphical representation, can introduce significant computational difficulties. These loops complicate the efficient contraction of tensors, a process essential for calculating physical quantities such as ground state energies or expectation values. In contrast to loop-free ansätze, where tensor contractions can be performed in a straightforward, hierarchical manner, the presence of loops often leads to an exponential increase in computational cost. This issue becomes particularly pronounced in higher-dimensional systems, where PEPS is most commonly applied.

A crucial challenge lies in the spurious entanglement that may arise around a loop, effectively increasing the Hilbert space dimension of the links and leading to inefficient manipulations of the networks. The elimination of these false correlations would allow for the efficient contraction of loopy tensor networks by transforming them into their unitary form and reducing the link dimensions.

This thesis addresses the challenge of removing spurious entanglement in loopy tensor networks and aims to develop a novel and efficient algorithm applicable to loopy tensor network ansätze. This development will enable the effective implementation of the unitary gauge in loopy tensor networks, leading to more efficient contractions and accurate simulations.

The thesis presents the following structure:

- Chapter 1

  Provides a review of numerical methods for investigating quantum many-body systems and introduces Tensor Networks, highlighting the differences between loopless and loopy tensor networks.

- Chapter 2

  Introduces the framework of loopy tensor networks and details the implementation of the proposed algorithm, comparing direct search methods with an AI-based neural network approach.

- Chapter 3

  Presents and discusses the results, focusing on the challenges of achieving efficient and high-quality solutions.

- Conclusion

  Summarizes the findings and outlines possible directions for future research.

Chapter 1

Numerical simulation of many-body quantum systems

Quantum Many-Body (QMB) systems present an intrinsic difficulty in computational simulation due to the exponential growth of the configuration space with the number N of elementary constituents (particles, lattice sites, etc.). For N elements with d possible states $\{\alpha_i\}_{i=1}^{N}$ (spin configurations, orbitals, energy levels, etc.), the number of possible configurations for the system is d^N . In quantum mechanics, this complexity is faced once one evaluates the fundamental object of interest, the wave-function $\psi(\alpha_1, ..., \alpha_N)$, whose modulus square gives the probability of the system being in the N-body configuration. Currently, only a few models can be solved exactly, and exact diagonalization techniques [6], required to solve the quantum mechanical problem, are restricted to small-sized systems, leaving the thermodynamic limit (large N limit) intractable via exact approaches.

In order to address this problem, various approximate analytical and numerical techniques have been developed. Semi-classical treatments of quantum degrees of freedom [7], such as mean-field approaches, e.g. the Hartree-Fock method [8] and Density Functional Theory [9], succeed in finding solutions. However, these methods, while highly performing in high spatial dimensions, perform poorly in low-dimensional systems [10] (e.g., 1D) and systems where entanglement and quantum fluctuations play crucial roles (e.g. Topological systems [11, 12], FQH [13], Spin liquids [14], etc.). Other successful approaches relie on Monte Carlo methods [15, 16], which are stochastic techniques for simulating QMB systems at equilibrium. These methods capture the statistical and quantum content of the system by reconstructing the model partition function. Unfortunately, complexities arise whenever a negative or complex action appears (the sign problem [17]). An alternative approach involves numerical methods based on the Renormalization Group (RG) paradigm. The core idea is to truncate the exponentially increasing Hilbert space based on energy considerations, as the low-energy physics of a system is primarily determined by its low-energy sectors [18]. In RG algorithms, the system size is then iteratively increased, and at each iteration, only low-energy sectors are retained. This framework has been further extended with the use of the density matrix, leading to the development of the Density Matrix Renormalization Group (DMRG) algorithm [1], where the truncation of the Hilbert space through a renormalization step is based on the population of the density matrix, with low-populated states being discarded.

Within this framework, Tensor Network (TN) methods have been developed [19]. TNs are a computationally manageable variational ansatz capable of capturing the many-body properties of a system with a controlled level of approximation based on the used resources [20, 21, 22, 23, 24, 25, 26]. Due to the tensor structure of the Hilbert space, although the exact amplitude of the system may be represented as a fully general rank-N tensor, much smaller, excellent approximations can be employed to build an efficient representation. The rank-N tensor can be indeed decomposed into a network of lower-rank tensors, which are equivalent up to the introduced approximations. By definition, the complexity of these states is directly related to the entanglement present in the system: TN states can span the whole quantum manifold constrained by entanglement bounds [19, 27]. In significant cases,

like short-range Hamiltonians, entanglement follows a scaling law known as the *Area Law* of entanglement [28, 29, 30, 31, 32, 3], allowing for the introduction of TN states with different geometries and topologies (TN ansätze) based on these tight constraints on entanglement content. As a consequence, TNs states provide an efficient representation when the QMB system under investigation displays sufficiently low entanglement. It is worth noting that once a tensor network ansatz is introduced, the network geometry can be manipulated to benefit algorithms; the dimensions of the tensor indices and the elements of each tensor are variables to be optimized to construct the most efficient representation of the system.

1.1 Tensor networks

Tensor Networks can be employed in many ways for quantum problems. In this field of physics, the approach is the Tensor Network ansatz state, which aims at expressing the huge complex amplitudes' tensor with a manageable amount of variables encoded as a Tensor Network.

Consider a QMB system, the wavefunction $|\psi_{QMB}\rangle$ lives in the N-body Hilbert space $\mathcal{H}_N$ defined by the tensor product of the all local ones $\mathcal{H}_N = \bigotimes_{i=1}^{N} \mathcal{H}_i$. The wavefunction provides the probability amplitude for the system to be in the N-body configuration $|\alpha_1 ... \alpha_N\rangle$, where α_i represents the single-body configuration of the i-th lattice site characterized by d possible states, $\{\alpha_i^j\}_{j=1}^{d}$. A generic state is thus described by the rank-N tensor:

$$|\psi_{QMB}\rangle = \sum_{\alpha_1 ... \alpha_N} T_{\alpha_1 ... \alpha_N} |\alpha_1 ... \alpha_N\rangle \tag{1.1}$$

We can represent tensors using the diagrammatic representation: a tensor with rank k is represented as a ball with k outgoing links (Fig. 1.1).

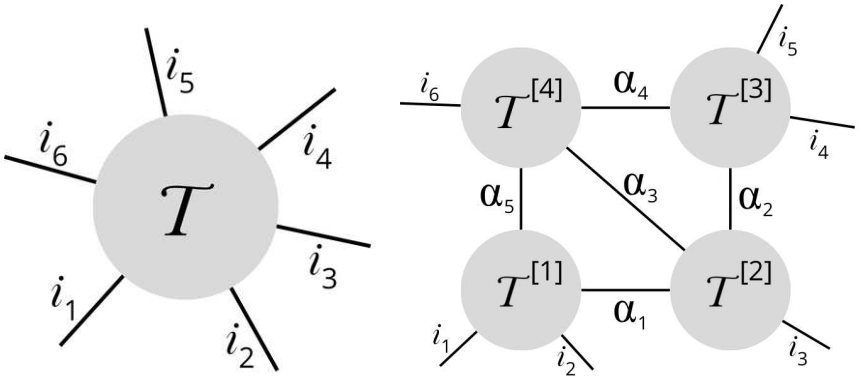

Figure 1.1: *Left*: QMB tensor with $N = 6$ outgoing links. The tensor is represented by a round object with open links representing physical indexes. *Right*: tensor network, i.e. a possible decomposition of the QMB tensor with $Q = 4$ nodes and $L = 5$ auxiliary links. The original tensor is recontructed by contracting over the auxiliary indexes. [33]

A tensor network is a network composed by a set of smaller tensors $T^{[q]}$, connected via the contraction over auxiliary (virtual) indices (Fig. 1.1):

$$T_{\alpha_1 ... \alpha_N} = \sum_{\beta_1, ..., \beta_L} T^{[1]}_{\{\alpha\}_1, \{\beta\}_1} ... T^{[Q]}_{\{\alpha\}_Q, \{\beta\}_Q} \tag{1.2}$$

for a network with Q nodes and L links. Each tensor $T^{[q]}$ can possess any number of physical indices $\{\alpha\}_q = \{\alpha_s : s \text{ physical index at node } q\}$ and auxiliary indices $\{\beta\}_q = \{\beta_\eta : \eta \text{ auxiliary index at node } q\}$. In order to build a TN computationally efficient for representing quantum many-body states, the number of links should not scale with the system size N and the auxiliary dimensions $D_\eta = \#\{\alpha_\eta\}$ (where the symbol # denotes the cardinality of a set) should be as well non-scaling quantities. The number of tensors Q instead scales with N.

TN ansätze allow to extract relevant informations of the system:

- State norm $\langle\psi_{QMB}|\psi_{QMB}\rangle$

  Evaluating the norm of the tensor network in polynomial computational time as a function of the number of tensors is referred to as *efficient contractibility*. This property is not guaranteed in all TN ansätze: loop-free TNs, such as MPS and TreeTN, possess this property, while other ansätze, like loopy tensor networks (PEPS, etc.) do not. In general, the calculation of the state norm can be made cheaper by exploiting the use of TN gauges rules.

- Overlap $\langle\psi_{QMB}|\psi'_{QMB}\rangle$

  This quantity shares the same numerical cost of the state norm, providing the states $|\psi_{QMB}\rangle$, $|\psi'_{QMB}\rangle$ in the same TN geometry and the resulting tensor networks to be efficiently contractible. Unfortunately, in this quantity, TN gauge invariance cannot be used.

- Expectation values of tensor product observables $\langle\psi_{QMB}|\bigotimes_s O^{[s]}|\psi_{QMB}\rangle$

  For homogeneous or site-dependent single-site operator $O^{[s]}$, efficient evaluation of expectation values is guaranteed by the possibility of absorbing the operators in the TN states, without altering the geometry of the network and thus preserving the efficient contractibility. Subclasses of these observables are local observables, correlation functions, string observables.

- Expectation values of operators in tensor network form

  For observables expressed in TN formalism, the expectation value is an efficient contractible operation.

- Entanglement properties

  Bipartitions of the tensor networks correspond to bipartitions of the physical system. It is then possible to reconstruct the spectrum of the reduced density matrix and thus the von Neumann and Rényi entropy, by contracting either of the two sub-networks.

1.1.1 Tensor manipulations

In a tensor network, data are organized in tensors, which in turn are represented by floating-point numerical data in the memory of the computer (RAM, hard drive, etc.). The way data are stored is fundamental in order to perform efficiently linear algebra operations. In this section, tensor operations are introduced to efficiently manipulate, compress and transform the entries of the tensors within the tensor network.

A tensor T with $n \in N_0$ links is a n-dimensional array $T_{\alpha_1,...,\alpha_N}$ of complex valued elements. We can define a mapping:

$$T : \mathbb{I}_1 \otimes ... \otimes \mathbb{I}_N \to \mathbb{C} : (\alpha_1, ..., \alpha_N) \to T_{\alpha_1,...,\alpha_N} \tag{1.3}$$

where the entries at position $(\alpha_1, ..., \alpha_N)$ are accessed through a tuple of n integers. The index α_i at position i takes values in the index set $\mathbb{I}_i = 1, ..., d_i \subset N$ and is referred to as the i-th link of the tensor. At the level of basic operations for links and tensors, it is not required to distinguish between auxiliary and physical links. This distinction is performed at the level of the tensor network.

Following the standard programming, tensor elements are stored in contiguous sectors of linearly addressed computer memory (array). Each element $T_{\alpha_1,...,\alpha_N}$ can be stored and addressed in such a linear array by assigning a unique integer index (address offset) to it. There are many ways of implementing this storing model. Due to efficient coding requirements, one should consider the advantages of these techniques and the programming languages as well, since different languages and compilers use intrinsic storing methods. A simple, but effective, way to implement the storing is "column-major" ordering:

$$\text{offset}(\alpha_1, ..., \alpha_N) = \sum_{i=1}^{N} (\alpha_i - 1) \prod_{k=1}^{i-1} d_i \tag{1.4}$$

To work effectively with tensor networks and develop efficient algorithms, it is essential to perform basic tensor manipulations. These fundamental operations involve adjusting elements or the structure of individual tensors, including their positions, number, and dimensions of links. Such manipulations often require reallocating and reorganizing data within computational memory.

Since tensor network algorithms frequently operate at the tensor level, they spend a substantial amount of their runtime on these basic operations. Therefore, it is crucial for numerical implementations to ensure that these operations are highly optimized. This often involves employing robust and efficient linear algebra libraries, such as the Linear Algebra PACKage (LAPACK) [34] and the Basic Linear Algebra Subprograms (BLAS) [35].
Introduce some basic tensor manipulations:

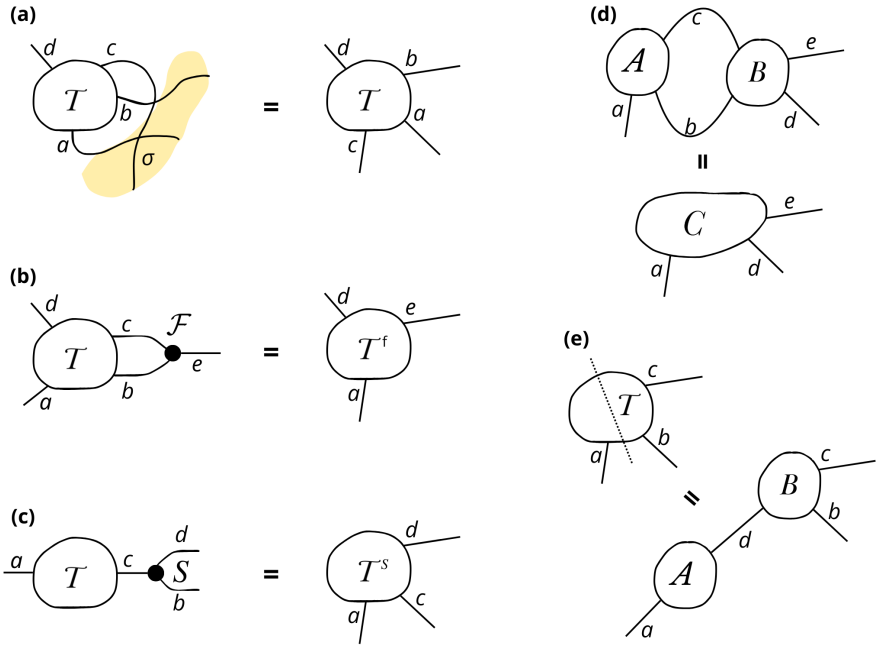

Figure 1.2: Tensors manipulations: (a) Link permutation. Intertwine the links, shown in the shaded area σ. (b) Link fusion. It can be seen as a contraction with a fuse-node F. (c) Link splitting. It can be seen as a contraction with a split-node S. (d) Tensor contraction. (e) Tensor decomposition. [33]

Link Fusion and Splitting

A rank-N tensor can be rearranged into a tensor of different rank following an invertible rule, such as a d-nary coding of a global index β $(\alpha_1, ..., \alpha_N = \text{d-coding}(\beta))$.

The **link fusion** involves combining $m - k + 1$ adjacent links at position $k, ..., m$ into a single link $\beta \sim \alpha_k, ..., \alpha_m$:

$$T'_{\alpha_1,...,\alpha_{k-1},\beta,\alpha_{m+1},...,\alpha_N} = T_{\alpha_1,...,\alpha_{k-1},(\alpha_k,...,\alpha_m),\alpha_{m+1},...,\alpha_N} \tag{1.5}$$

The new index takes values in the Cartesian product of all fused links.

The **link splitting** is the inverse operation of the link fusion. This operation returns a tensor with a link β replaced by $m - k + 1$ links at positions $k, ..., m$, so that the product of the dimensions $d_{\alpha_k}, ..., d_{\alpha_m}$ equals the dimension d_β.

Both operations, link fusion and splitting, can be realized at a negligible numerical cost $o(1)$.

Tensor Contraction

The tensor contractions are a generalization of matrix multiplication. Consider two tensors A, B with links $\{\alpha_1, ..., \alpha_n\}$, $\{\beta_1, ..., \beta_m\}$ of dimensions $\{d^{\alpha}_{1,...,n}\}$, $\{d^{\beta}_{1,...,m}\}$ and suppose they have a subset of k

common links $s_j = \{\alpha_j\}_{j=n-k}^{n} = \{\beta_j\}_{j=1}^{k}$. The contraction over these common links is the contracted tensor C spanned by the remaining links $n + m - 2k$:

$$C_{\alpha_1,\ldots,\alpha_{n-k},\beta_{k+1},\ldots,\beta_m} = \sum_{s_1,\ldots,s_k} A_{\alpha_1,\ldots,\alpha_{n-k},(s_1,\ldots,s_k)} B_{(s_1,\ldots,s_k),\beta_{k+1},\ldots,\beta_m} \tag{1.6}$$

Usually, before the contraction, links are not ordered and link permutation is required to match the correct subset of the indexes. The computational complexity of this operation scales as the product of all the dimensions of the tensor links involved, considering the contracted ones only once, thus it scales as $o(d_\alpha d_s d_\beta)$.

This tensor contraction can be performed more efficiently by recasting the two tensors A, B in a matrix form. Indeed, one can take advantage of parallelization and use optimized linear algebra tools for matrix multiplication:

$$C_{a,b} = \sum_l A_{a,l} B_{l,b} \tag{1.7}$$

However, one must consider the overhead produced by the required permutations, thus a careful analysis must be performed to build the best efficient contraction possible.

Tensor Decomposition

The tensor decomposition allows to transform an initial tensor into two or more tensors, attached together through additional internal links. The contraction over these internal links must reproduce the original tensor with a certain precision. This operation is advantageous whenever the output tensors present some properties, such as smaller sizes and (or) interesting isometries. The most important tensor decompositions belong to a class of matrix-factorizations: the SVD-decomposition and the QR-decomposition.

Both manipulations require to recast the input tensor in a matrix form, thus merging two bipartitions of the links into two global indices (if required, link permutation has to be performed beforehand). The matrix is then decomposed into factors matrix and, eventually, the fused links are split to match the original network geometry.

The **SVD decompositions** reads:

$$T_{(i_1,\ldots,i_q),(i_{q+1},\ldots,i_n)} \overset{\text{fuse}}{=} T_{l,r} \overset{\text{SVD}}{=} \sum_k^{d^k} \mathcal{V}_{l,k} S_{k,k} \mathcal{W}_{k,r} \overset{\text{split}}{=} \sum_k^{d^k} \mathcal{V}_{(i_1,\ldots,i_q),k} \lambda_k \mathcal{W}_{k,(i_{q+1},\ldots,i_n)} \tag{1.8}$$

It produces a diagonal and positive real matrix S, where the real valued numbers λ_k are called singular values, and two semi-unitary (or isometric) matrices $\mathcal{V}$, $\mathcal{W}$ with respect to k. The semi-unitary matrices obey to $\mathcal{V}^\dagger\mathcal{V} = \mathbb{1}$, $\mathcal{W}\mathcal{W}^\dagger = \mathbb{1}$, while it does not necessarily hold for $\mathcal{V}\mathcal{V}^\dagger$ and $\mathcal{W}^\dagger\mathcal{W}$.

This decomposition presents some advantages:

- The singular values reveal entanglement properties, in particular in loop-free tensor networks.
- Singular values equal to numerical zeros can be neglected, thus reducing the dimensions of the links.
- Small singular values can be approximated to zeros, again effectively reducing the links dimensions, but introducing an error which scales with the truncation threshold ϵ.
- The semi-unitary tensors, due to their particular structure, allow to establish network gauges.

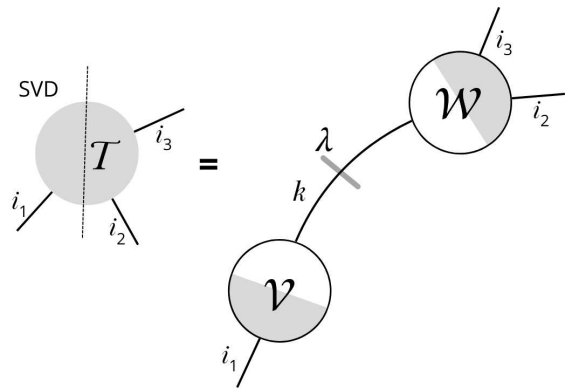

Figure 1.3: SVD-decomposition of a tensor T produces an intermediate link k and two semi-unitary tensors $\mathcal{V}$ and $\mathcal{W}$ (depicted by shade over the orthonormal link bipartition, see also Fig. 1.4). The singular values matrix S is represented by a bar to highlight the sparsity. Compressing the dimension d_k by neglecting some singular values λ_k does not disturb the tensor isometries. [33]

The **QR-decomposition** reads:

$$T_{(i_1,\ldots,i_q),(i_{q+1},\ldots,i_n)} \overset{\text{fuse}}{=} T_{l,r} \overset{\text{QR}}{=} \sum_k^{d^k} Q_{l,k} R_{k,r} \overset{\text{split}}{=} \sum_k^{d^k} Q_{(i_1,\ldots,i_q),k} R_{k,(i_{q+1},\ldots,i_n)} \tag{1.9}$$

It produces an upper-trapezoidal matrix $R_{k,r}$ and a semi-unitary (or isometric) matrix $Q_{l,k}$ with respect to k, so that $\sum_l Q_{l,k} Q^*_{l,k'} = \delta_{k,k'}$.

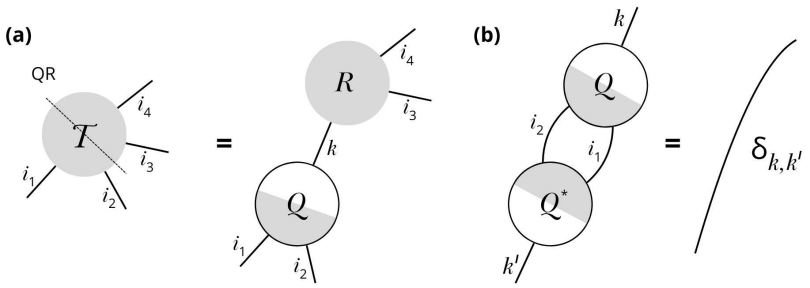

Figure 1.4: QR decomposition and isometry. (a) QR-decomposition of a tensor T into an upper-trapezoidal matrix R and an isometry Q. (b) The isometry is depicted by a gray-white filling: the gray partition indicates the links over which we have unitarity. [33]

Both decompositions display a computational complexity of $o(d_{max} \cdot d_k^2)$, where $d_{max} = \max\{d_1 \cdot \ldots \cdot d_q, d_{q+1} \cdot \ldots \cdot d_n\}$ is the maximum dimension of the fused links.

Tensor Compression

Tensor compression allows to reduce the link dimensions of the tensor. Considering a rank-N tensor T, through link fusion it is possible to recast the tensor in a matrix form and thus it is now possible to exploit the linear algebra potential acting on matrices. Indeed, a SVD decomposition can act over the matrix to decompose it into three tensors:

$$T_{i,j} = \sum_k \mathcal{V}_{i,k} S_{k,k} \mathcal{W}_{k,j} \tag{1.10}$$

where S is a diagonal and positive real matrix, while $\mathcal{V}$,$\mathcal{W}$ are unitary matrices (or isometries). Since S is positive and real, its diagonal elements can be ordered, specifically in ascending order, and they are bounded from below by zero. The crucial fact is that if $\lambda_i < \epsilon$ for $\forall i > j$, one can discard some

singular values: whenever the singular values decay fast, one can neglect them and keeping the first m. This operation usually introduces some errors, estimated by:

$$||T - \sum_{k=1}^{m} \mathcal{V}_{i,k} S_{k,k} \mathcal{W}_{k,j}|| = || \sum_{k=m+1}^{d^N} \mathcal{V}_{i,k} \lambda_k \mathcal{W}_{k,j}|| < || \sum_{k=m+1}^{d^N} \mathcal{V}_{i,k} \epsilon \mathcal{W}_{k,j}|| < \epsilon C \tag{1.11}$$

where C is a finite constant. Note that if ϵ decay fast to zero, or to its numerical representation of zero, this approximation introduced no errors, but allows to reduce drastically the tensor dimensions (to m), keeping the used resources small enough.

The inverse operation of tensor compression is tensor inflation, also known as padding, where the link dimensions are instead enlarged.

Tensor network gauge

Tensor network states usually contain some redundant information that lead to inefficient algorithms. It is then possible to manipulate the tensor network with a set of linear transformations, which leave the quantum state and its physically relevant quantities unchanged. These operations are referred to as *gauge transformations*. The *gauge invariance* is based on the fact that local invertible operations acting on the auxiliary TN spaces do not influence the physical degrees of freedom.

Consider two tensors $T^{[q]}$, $T^{[q']}$ at node q, q' contracted together through an auxiliary index η with link dimension D_η and recast in matrix form. It is possible to introduce on the virtual link the equivalence $\mathbb{1} = U^\dagger U$ for a unitary matrix U:

$$T^{[q]}_i T^{[q']}_j = T^{[q]}_i U^\dagger_{i,k} U_{k,i} T^{[q']}_j = \tilde{T}^{[q]}_k \tilde{T}^{[q']}_k \tag{1.12}$$

or in general for a $D'_\eta \times D_\eta$ (left-) invertible matrix $Y_{\alpha_\eta, \alpha_{\eta'}}$ ($\sum_{\alpha'} Y^{-1}_{\alpha,\alpha'} Y_{\alpha',\alpha''}$), if the tensors transform as:

$$T^{[q]}_{\{i\},\{\alpha\}_{\setminus\eta},\alpha_\eta} \to \tilde{T}^{[q]}_{\{i\},\{\alpha\}_{\setminus\eta},\alpha_\eta} = \sum_{\beta_\eta} Y_{\alpha_\eta,\beta_\eta} T^{[q]}_{\{i\},\{\alpha\}_{\setminus\eta},\beta_\eta} \tag{1.13}$$

$$T^{[q']}_{\{i'\},\{\alpha'\}_{\setminus\eta},\alpha_\eta} \to \tilde{T}^{[q']}_{\{i'\},\{\alpha'\}_{\setminus\eta},\alpha_\eta} = \sum_{\beta_\eta} Y^{-1}_{\beta_\eta,\alpha_\eta} T^{[q']}_{\{i'\},\{\alpha'\}_{\setminus\eta},\beta_\eta} \tag{1.14}$$

then one can change the tensor entries to exploit some favourable relations, while leaving the tensor network geometry unchanged.

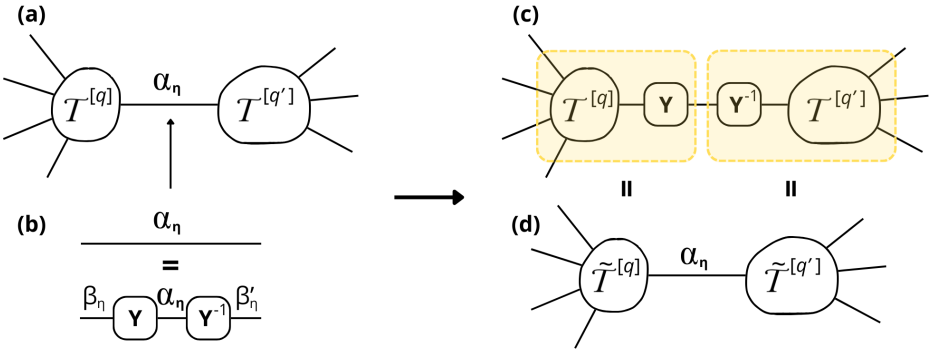

Figure 1.5: Gauge transformation. Consider a tensor network (a), introduce on the virtual link α_η between tensors $T^{[q]}$ and $T^{[q']}$ the equivalence of the matrix Y and its inverse (b). Contract Y and Y^{-1} into their respective neighboring tensors (c) and obtain an alternative description of the tensor network, without changing the physical content of the stored information. [33]

This relation can be exploited at the level of fused links, as well at the level of single indexes. These algebraic manipulations drastically reduce the computational cost of global contractions, allowing to design efficient algorithms which operate on the whole tensor network.

It is worth noting that this operation is physically different from gauge invariant tensor networks. Indeed, despite the similar name, the two constructions are in general different and have different physical origins. In the former case, one takes advantage over the invariance under transformations on auxiliary indexes, while in the latter the invariance arises from the symmetries of the theory one is studying and the gauge transformations act on the physical indexes.

1.1.2 Loopless tensor networks and DMRG

General aspects

Tensor networks that do not contain cycles are commonly referred to as *loopless tensor network*. The most important examples of these networks are the Matrix Product State (MPS) ansatz and the Tree Tensor Network (TTN) ansatz. Loopless tensor network can exploit the whole potential of gauge transformations, effectively reducing the number of contractions and thus improving the computational speed. This favourable property allows to work with higher link dimensions and may compensate for the lower entanglement description, with respect to the loopy tensor network framework.

The loopless tensor network structure inherently defines a distance $\mathrm{dist}(a, n)$ between any two nodes a, b in the network, which represents the number of links encountered in the shortest path connecting the nodes. All the nodes that share the same distance d with regard to a central node c belongs to the level $L(d, c)$. Note that in loop-free tensor network, the shortest path is unique. Besides this property, one can introduce a maximal virtual link dimension $D = \max_\eta(D_\eta)$, which is generally larger than the physical dimension d_s and it is fundamental in all tensor networks. The tensor network contraction computational complexity, starting from the outermost level towards the center node, is a scaling function of the dimension D and the maximal number M of links of a tensor in the network and thus it is upper-bounded by $o(D^{M+1})$. In specific network geometries, tighter bounds can be implemented, e.g. in MPS networks where each node carries a physical dimension and thus the general complexity scales differently.

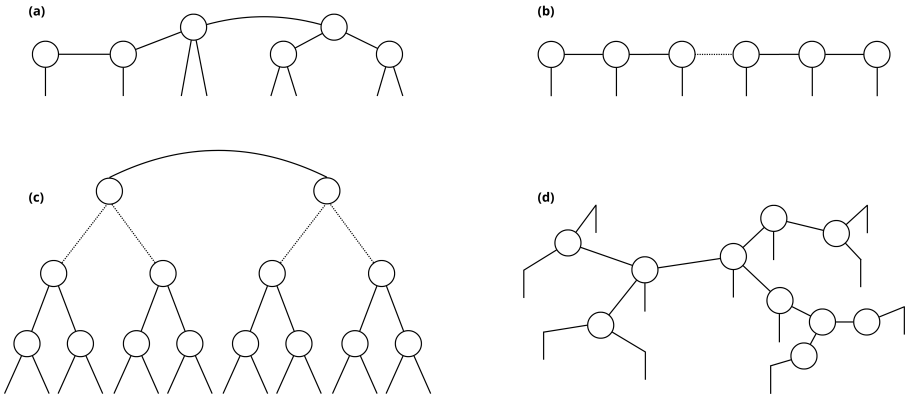

Figure 1.6: Different loop-free TN geometries: (a) a generic tree tensor network, (b) an MPS, (c) a binary tree tensor network (bTTN), (d) a molecular TTN, shaped according to an arbitrary loop-free graph geometry. [33]

Another fundamental property of loopless tensor networks is that every link between any two nodes induces a bipartition of the system. Considering a virtual link η, we can rewrite the tensor network through a Schmidt decomposition:

$$|\psi\rangle = \sum_{k=1}^{D_\eta} \lambda_k \, |A_k\rangle \, |B_k\rangle \tag{1.15}$$

where the states $|A_k\rangle$, $|B_k\rangle$ describes the two partition respectively and the numbers λ_k are the Schmidt values (we can assume them to be sorted in descending order). Based on this, it is possible to define some properties:

- A loopless tensor network can be seen as a simultaneous decomposition over all virtual links.

- The bond dimension D provides an upper bound for the Schmidt rank and thus the bipartite entanglement.
- Optimal single link compression is achieved by discarding (setting to zero) the smallest singular values (subsect.1.1.1). Compression of the dimension D_η to χ_η is obtained by truncating the summation at $k = \chi_\eta$ and neglecting the smallest singular values:

$$|\psi_{trunc}\rangle = \frac{1}{N_{kept}} \sum_{k=1}^{\chi_\eta} \lambda_k |A_k\rangle |B_k\rangle \quad , \quad N_{kept} = \sqrt{\sum_{k=1}^{\chi_\eta} \lambda_k^2} \tag{1.16}$$

  According to the Eckart–Young–Mirsky theorem, this procedure leads to the smallest error in quantum state fidelity. Performing this compression on multiple links simultaneously instead increases the error and does not provide the optimal quantum state fidelity.
- Detaching an n-link network node from a loop-free TN induces a partition into n disjunct subspaces. Each virtual index then contributes variational degrees of freedom in form of associated tensor elements.

Gauging

In the framework of loop-free tensor network, one has the freedom to implement specific gauges that are related to the orthogonality properties of the tensors in the network. In particular, the unitary gauge and its refinement, the canonical gauge, allow to install isometries in each tensor, which tremendously simplify the contractions in the network with its conjugate. They also enable efficient local error estimation and efficient application of local operators in time evolution algorithms and measurements. Furthermore, in variational algorithms, they maintain the network state's normalization during local optimization.

Conceptually, the canonical gauge provides an understanding of the loop-free TN ansatz as simultaneous, independent, truncated Schmidt decompositions over all lattice bipartitions induced by the network geometry.

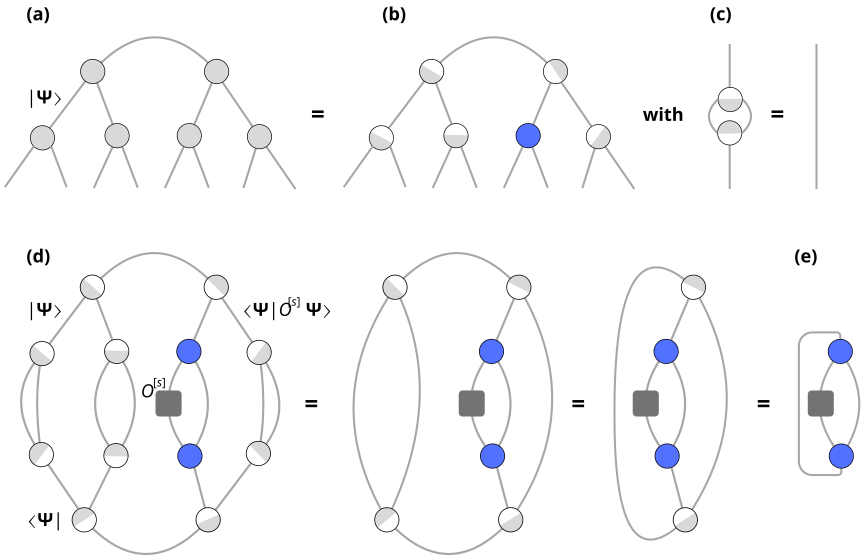

Figure 1.7: Example of efficient contraction on a binary tree TN (a) describing a QMB state $|\Psi\rangle$. Installing a specific gauge (b), imposes some rule (c) on the TN and drastically simplifies numerical calculations, e.g. the calculation of the expectation value of a local observable $\langle\Psi| O^{[s]} |\Psi\rangle$. The bra TN is drawn as the vertically flipped version of the ket TN, indicating hermitian conjugation of all its tensors (d). Thanks to the installed gauge, most contractions do not have to be performed explicitly numerically, while it would have been necessary otherwise. In the end, only the contractions that can not be simplified remain to be executed (e), leading to a much geometrically simpler tensor network. [33]

- **Unitary gauge**

The unitary gauge prescription requires a single center node c to be selected. The result tensor network is said to be in *unitary form*.
Starting from the outermost level, the nodes with maximal distance $d = d_{max}$ from the center, the unitary gauge is installed on each level $\{L(d, c) : d = 0, ..., d_{max}\}$ following these steps (Fig. 1.8):

1. For each node $q \in L(d, c)$, perform the QR-decompositon:

$$T^{[q]}_{\{s\},\{\alpha\},\alpha_\eta} = \sum_{\beta_\eta=1}^{\tilde{D}_\eta} Q^{[q]}_{\{s\},\{\alpha\},\beta\eta} R^{[q]}_{\beta_\eta,\alpha_\eta} \tag{1.17}$$

   on the bipartition between the virtual link η that leads towards the center c and all the remaining tensor links of q, the physical $\{s\}$ and the virtual $\{\alpha\}$ ones.

2. Update the tensor $T^{[q]} \to \tilde{T}^{[q]} = Q^{[q]}$ with the semi-unitary isometry of the QR decomposition

3. Contract the upper trapezoidal matrix $C^{[\eta]} = R^{[q]}$ over the corresponding virtual link η into the adjacent tensor from the inner layer $L(d-1, c)$.

4. Proceed to the next inner layer $d \to d-1$ and repeat from step 1 until d = 0 is attained.

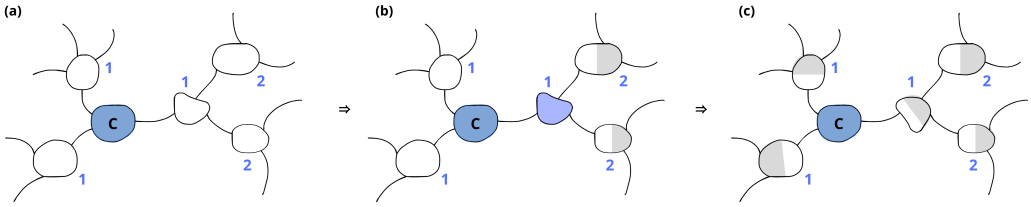

Figure 1.8: Example for a loop-free TN gauging procedure. The numbers indicate the level of each node, determining the order of isometrization. Starting from the un-isometrized network (a), tensors with largest distance are isometrized first, and the next tensors towards the center are updated (b). Then the isometrization continues on the next level (c). This procedure is repeated until the center (blue) is reached. [33]

All tensors have been updated to $\tilde{T}^{[q]}$. Also, note that virtual links might turn out reduced in bond dimension from D_η to $\tilde{D}_\eta$, since the QR-decomposition selects the minimum between D_η and the product of the remaining link dimensions. This procedure implements a global gauge transformation, since the physical state of the network remains unchanged. The computational complexity is dominated by the QR factorizations upper-bounded by $o(D^{M+1})$, with D maximal bond dimension and M maximal number of links of a tensor.

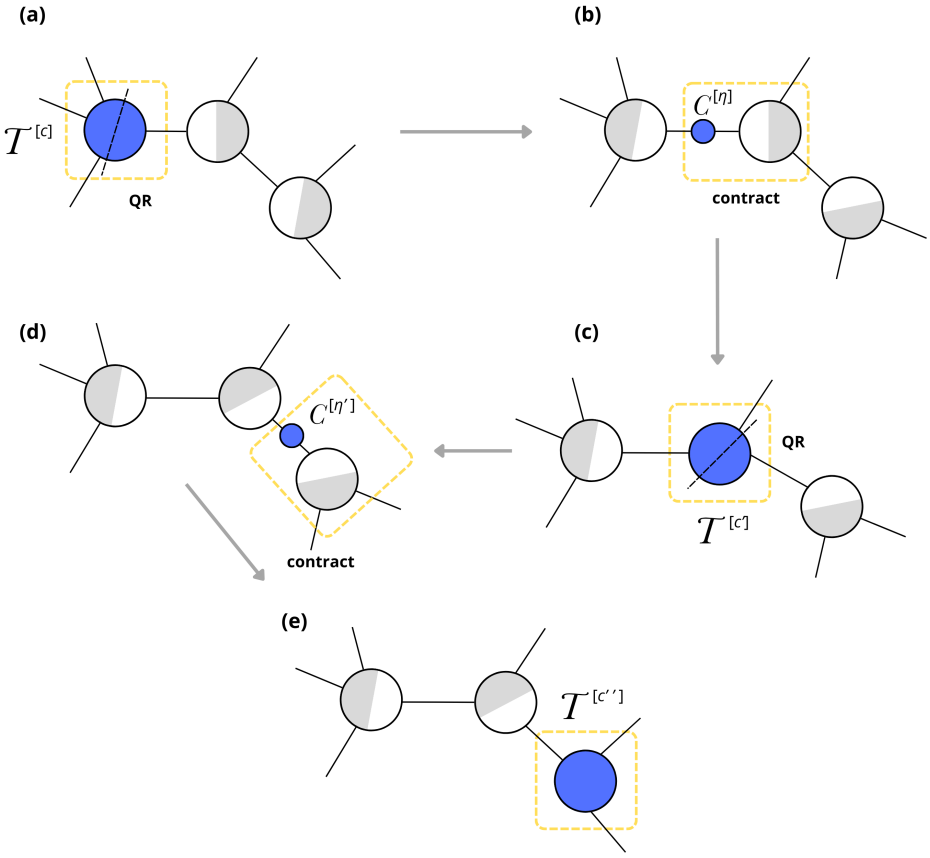

Figure 1.9: Moving the center c in unitary gauge to a network node c''. Network center-tensors on the path of links η, η' between the old and the new selected center node are QR-decomposed (a,c) and the non-unitary parts $C^{[\eta]}$, $C^{[\eta']}$ are subsequently contracted into the respective next node on the path (b,d). Tensor isometrization (gray-white shading) is always directed to the current gauge center. [33]

The next step is to move the center node from a node c_1 to a node c_2 (Fig. 1.9). This operation is performed with a sequence of QR-decompositions $T^{[q]} = \tilde{T}^{[q]} C^{[\eta]}$ of all nodes q over the path connecting c_1 and c_2. In every step, a possibly different non-unitary part $C^{[\eta]}$ is passed over the shared bond η from the node $\tilde{T}^{[q]}$ to the next tensor on the path. This process yields the isometry relation of the unitary gauge (Fig. 1.10):

$$\sum_{\beta_\eta} T^{[q]}_{\{s\},\{\alpha\},\alpha_\eta} C^{[\eta]}_{\beta_\eta,\alpha_\eta} = \sum_{\beta_{\tilde{\eta}}} \tilde{T}^{[q]}_{\{s\},\{\alpha\},\alpha_{\tilde{\eta}}} C^{[\tilde{\eta}]}_{\beta_{\tilde{\eta}},\alpha_{\tilde{\eta}}} \tag{1.18}$$

where $C^{[\eta]}$ and $C^{[\tilde{\eta}]}$ are the (non-unitary) center matrices defined on any two links η and $\tilde{\eta}$ of q respectively, as they might be encountered on a path moving the center. This relation holds for every network node q and transforms the tensor $T^{[q]}$, which is isometric over η, into the tensor $\tilde{T}$, which is isometric over $\tilde{\eta}$.

If one have access to the single-sided inverses $(C^{[\eta]})^1$, the re-isometrization (Eq. 1.18)) can be performed with local contractions only, without costly matrix factorizations. However, the inversion requires the matrix C to be of full rank, and thus one needs to find a way to reduce the possibly oversized matrix dimensions first. The canonical gauge realizes such minimal bond dimensions, together with a very efficient re isometrization rule.

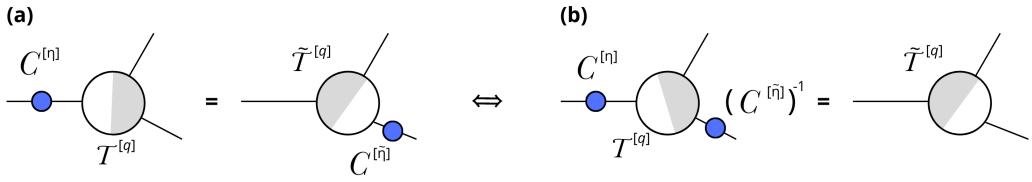

Figure 1.10: (a) Re-isometrization of a node in the unitary gauge (essentially steps (b)→(c) and (c)→(d) from Fig. 1.9), achieved by QR decomposition. (b) Formulation of the same transformation in terms of a link-local gauge transformation, by contracting the respective non-unitary center matrices $C^{[\eta]}$ and $(C^{[\tilde{\eta}]})^1$ (both blue) into their adjacent node (only possible if their (single-sided) inverses exist, i.e. if the bond dimensions equal the matrix ranks). [33]

- **Canonical gauge**

The canonical gauge is realized by SVD-decomposing each center matrices $C^{[q]}$ over all bonds η. The result tensor network is said to be in *canonical form*. The SVD-decomposition reads:

$$C^{[\eta]}_{\alpha_\eta,\beta_\eta} = \sum_{\gamma_\eta=1}^{\chi_\eta} \mathcal{V}^{[\eta]}_{\alpha_\eta,\gamma_\eta} \lambda^{[\eta]}_{\gamma_\eta} \mathcal{W}^{[\eta]}_{\gamma_\eta,\beta_\eta} \tag{1.19}$$

The isometries $\mathcal{V}^{[\eta]}$ and $\mathcal{W}^{[\eta]}$ act as link-local gauge transformations on η, as in Fig. 1.12.

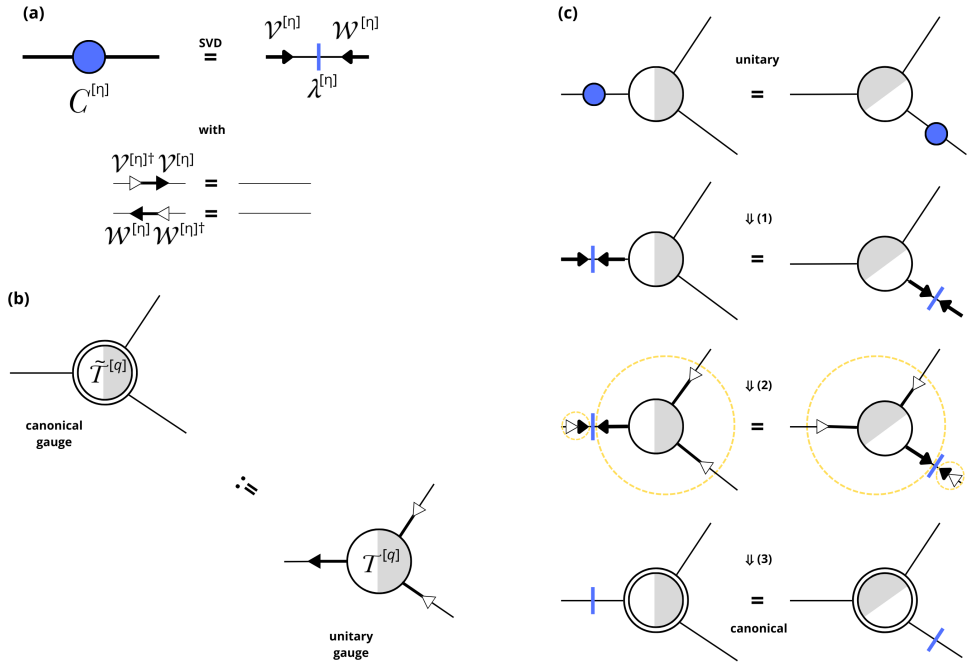

Figure 1.11: Transformation from unitary- to canonical gauge. (a) Define isometries $\mathcal{V}^{[\eta]}$, $\mathcal{W}^{[\eta]}$ (black triangles, pointing towards the center singular values) and their one-sided inverses (white) from left- and right- singular vectors of the unitary-gauge center matrices $C^{[\eta]}$ (blue) on all virtual links η in a TN. (b) The canonical-gauge tensor (double outline) is obtained from the unitary-gauge tensor by absorbing, from all virtual links, those isometries that point towards the current TN gauge center $C^{[\eta]}$ sitting on some network link η. Inverses are taken if tensor links lead away from the center. When repeated on each tensor, this procedure establishes the canonical gauge by insertion of link-local gauges $\mathcal{V}\mathcal{V}^\dagger$ or $\mathcal{W}^\dagger\mathcal{W}$ on each non-center virtual link (see Fig. 1.12), and $\mathcal{V}^{[\eta]}\lambda^{[\eta]}\mathcal{W}^{[\eta]}$ on the center link η. (c) The isometry relation of the canonical gauge (bottom) follows from the isometry relation in the unitary gauge (top): We (1) perform SVDs on the center matrices and (2) multiply all (virtual) links on both sides with the respective inverse isometries pointing towards the tensor node. (3) By exploiting (a) and (b), we recover an identity and the definition of a canonical tensor. The result (bottom) is the isometry relation in the canonical gauge picture. In order to visualize the compression to minimal bond dimension, entailed by the SVD, we assign different line widths to links of different dimension. [33]

The isometries $\mathcal{V}^{[\eta]}$, $\mathcal{W}^{[\eta]}$ are then absorbed into the adjacent nodes, thus updating the tensors $T^{[q]} \to \tilde{T}^{[q]}$ to the canonical tensors (Fig. 1.11). The singular values that are zero or fall below a certain

threshold are neglected, so that the virtual link bond dimensions are minimal, equal to the respective center-matrix ranks, $\chi_\eta \leq D_\eta$.

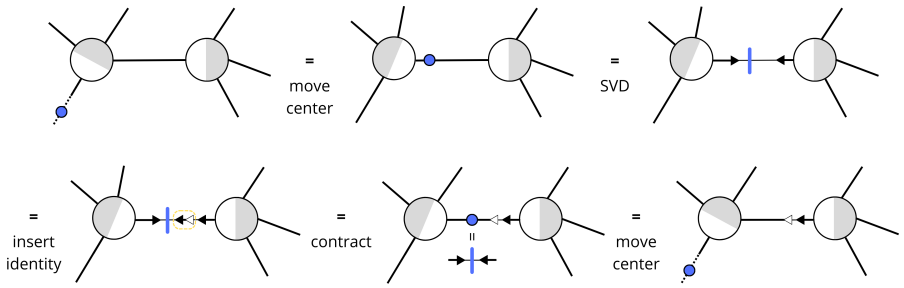

Figure 1.12: Pairs of isometries act indeed as link-local gauges in the transformation from unitary to canonical gauge. In unitary gauge, first move the center (blue) onto the selected virtual link (here depicted with two adjacent nodes as part of a larger TN) and compute its SVD (top left to right). The SVD yields a left-unitary term $\mathcal{V}$ and a right-unitary term $\mathcal{W}$ with $\mathcal{V}^\dagger\mathcal{V} = \mathcal{W}\mathcal{W}^\dagger = \mathbb{1}$. Then insert such an identity, restore the center matrix, and move it back to its original position. [33]

It is not required to find the center matrices $C^{[q]}$ on all bonds for installing this gauge. Indeed, installing a unitary gauge with a single center-bond η towards which all tensors are isometries is sufficient. Thereafter, to move the unitary center, a series of SVDs is performed in reverse order from center outwards, establishing the center matrix on all bonds (Fig. 1.13). The isometries induce orthonormality and thus every SVD of the center matrix achieves a Schmidt decomposition of the complete tensor network state, with Schmidt values $\lambda^{[\eta]}$ and Schmidt rank χ_η. In the end, the canonical gauge can be viewed as the explicit and simultaneous Schmidt decomposition of the QMB state on all lattice bipartitions induced by the loop-free network geometry. It is worth noting that, since the Schmidt decomposition is unique, then also the canonical gauge is unique. This gauge implements also the minimal bond dimension on all links and it is not possible to further reduce it without altering the state.

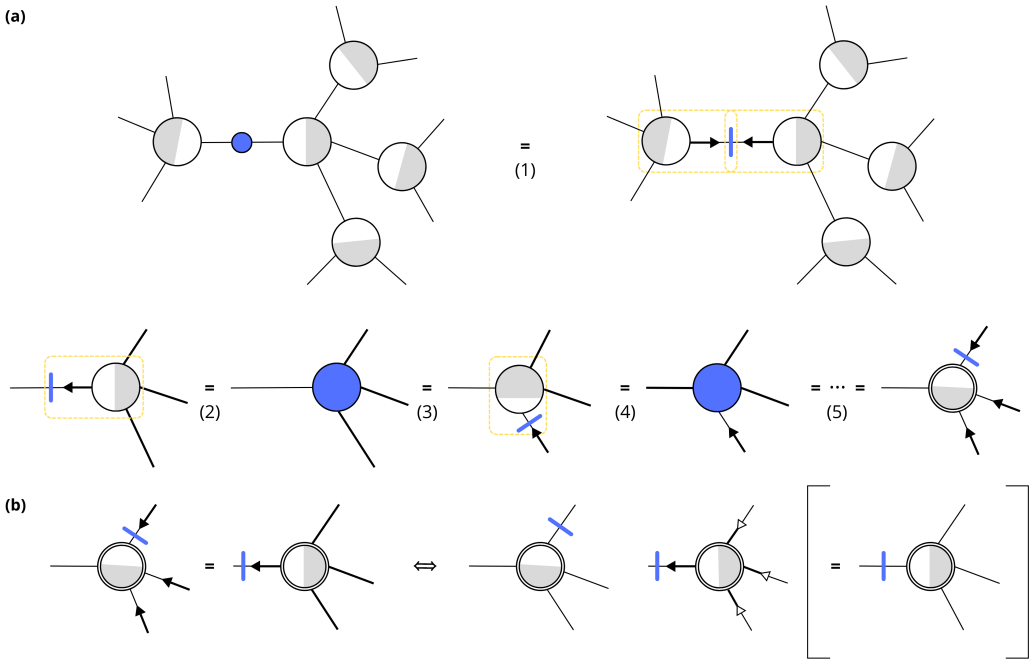

Figure 1.13: (a) Installation of the canonical gauge. (1) The gauge center (blue) of a TN in unitary gauge is SVD-decomposed. Starting with the nodes adjacent to the center, perform the following steps: (2) absorb unitary matrix and singular values from parent link, (3) perform SVD with respect to a first child link, (4) re-absorb singular values, (5) repeat previous two steps on remaining child links. The final tensor (without the unitaries on the child links) is now in canonical gauge. (b) Comparison of the final and the initial tensor allows to recover the definition of the canonical gauge. [33]

One of the fundamental property that makes this gauge so important is that the isometric structure of the gauged tensor network allows for efficient tensor network contraction. From this relation:

$$T^{[q]}_{\{s\},\{\alpha\},\alpha_\eta}\lambda^{[\eta]}_{\alpha_\eta} = \tilde{T}^{[q]}_{\{s\},\{\alpha\},\alpha_{\tilde{\eta}}}\lambda^{[\tilde{\eta}]}_{\alpha_{\tilde{\eta}}} \tag{1.20}$$

one can see that the tensor $T^{[q]}$, originally isometric over the bond η, turns into an isometry $\tilde{T}^{[q]}$ over another of its virtual links $\tilde{\eta}$ (Fig. 1.14). Because Schmidt values are easily storable and invertible, one can efficiently re-isometrize the tensor network using only local operations. The canonical gauge is then an advantageous choice in algorithms driven by (local) unitary evolution, e.g. real time evolution algorithms, because a unitary operation does not alter the Schmidt decomposition except for the bonds it acts on.

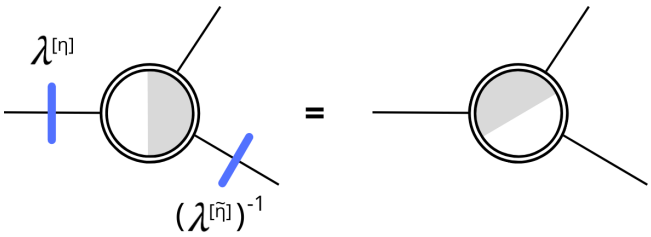

Figure 1.14: Re-isometrization of a node in the canonical gauge. By restoring the Schmidt values $\lambda^{[\eta]}$ (blue) for all links of a network, one can perform a re-isometrization of a node using only numerically efficient contractions with these "link-weights" and their inverses. Due to the minimal bond dimension in canonical gauge, the inversion is always possible. [33]

The Density Matrix Renormalization Group for loopless Tensor Networks

In this framework, **Density Matrix Renormalization Group (DMRG)**-based methods are easily implemented. DMRG is a powerful modification of the Renormalization Group (RG) algorithm, which is an approximated method based on the physical intuition that the ground state of a system is composed of low-energy states of the system's (non-interacting) bipartitions. Within this assumption, it is possible to introduce an algorithm that allows describing the ground state properties of many-body quantum systems with large sizes N, up to the thermodynamical limit corresponding to the fixed point of the renormalization flow. The DMRG enhances this technique by improving the truncation rule, leading to a higher precision description of the final state at the price of slowing down the growth of the system size. Indeed, in the DMRG algorithm, the system size increases linearly instead than exponentially with the number of iterations. The new truncation rule is based on keeping the states of the reduced density matrix of the QMB system bipartitions with the highest populations.

Ground state (GS) search algorithms, which are a natural extension of the DMRG algorithm, can be efficiently implemented in loop-free tensor network, as the absence of cycles leads to many computational advantages (e.g. the possibility to install the unitary gauge). In fact, the loop-free geometry allows to formulate local optimization problems (variational energy minimization procedures) as simple eigenvalue problems. The ground state search algorithm is a diagonalization problem, however, it is highly inefficient to attack with direct diagonalization routines. The solution is to split the problem into a set of smaller diagonalization problems formulated for a small enough subset of tensors. Once an initial random state is initialized, an iterative scheme of local tensor optimizations is employed to converge on the state that best approximated the true QMB ground state (Fig. 1.15). The algorithm is composed of several high-level operations ([33]).

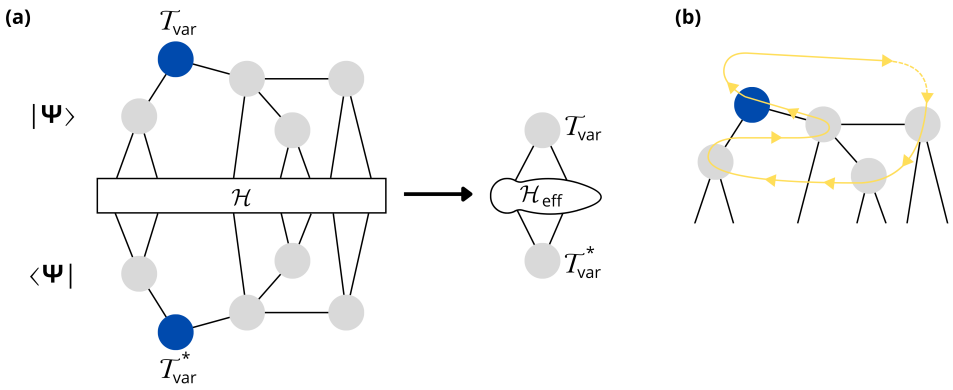

Figure 1.15: Basic idea of the variational ground state search. (a) In order to reduce the complexity of the minimization problem, only the parameters of one tensor T_{var} (blue) are taken as variational, while all other tensors are taken as fixed. The variational tensor forms the "optimization center", while the fixed tensors form an "environment". The latter can be efficiently contracted with the Hamiltonian H, leading to a reduced "effective" Hamiltonian H_{eff}. (b) By solving the reduced optimization problem sequentially (for varying locations of the optimization center), the energy expectation value is gradually reduced. This iterative minimization is commonly called "sweeping". [33]

Consider a loop-free tensor network ansatz composed of tensors $T^{[q]}$ with nodes $q = 1, ..., Q$, joined by virtual links η with dimension D_η. The unitary gauge is installed and one can arbitrarily choose any node to be the center node.

Setup

1. Definition of the tensor network structure:

   the TN has to be designed to comply best with the system and the Hamiltonian under investigation. The ansatz need to be compatible with the interaction terms in the Hamiltonian and needs to be capable of hosting the required content of entanglement, as well as its spatial distribution. The conditions that are meant to be matched are:

   - Tensor network with N physical links, hosting the Hamiltonian H of the system.
   - Tensor network with no cycles: there exists a unique path connecting any pair of nodes.
   - Virtual link dimensions must not exceed the bond dimension D of the TN.
   - In order to not generate issues with symmetries, avoid the use of tensors with less than three links.

2. Definition of the sweep sequence:

   this refers to the rule determining the sequential selection of optimization centers within the optimization loop. The exact form of the optimal sweep sequence, which achieves the fastest energy convergence in the shortest computation time, depends on the TN structure and the specific model being analyzed.

3. Initialization of the tensor network state:

   the TN state can be initialized randomly, which has the advantage of being less biased and reducing the risk of converging to a local energy minimum state. Another possibility is to initialize the state from an already computed ground state, with the same symmetries and quantum numbers.

4. Selection of the optimization center:

   - Install the unitary gauge on the network with respect to a center node c.
   - Construct an effective Hamiltonian H_{eff} for c.

Whenever a multi-tensor optimization is required, two or more network nodes can be chosen to be the network centers. This operation is achievable by temporarily contracting some adjacent node $\{q\}$ into a single center node c.

Optimization loop

1. Optimize $T^{[c]}$:

   the optimal $T^{[c]}$ is given by the eigenvector corresponding to the lowest eigenvalue of the effective Hamiltonian.

2. Select the next optimization center c':

   following the sweep sequence, move the isometry center to c' using gauge transformations and update the renormalized Hamiltonian terms between c and c' (Fig. 1.16).

3. Go back to step 1 and repeat the optimization with the new center node c', continue until the end of the sweep sequence is reached.

4. Ensure convergence:

   if the convergence is not reached, start a new sweeping sequence. The converge criteria are typically: change in energy expectation value, variance of energy, distance of TN states from subsequent sweeps.

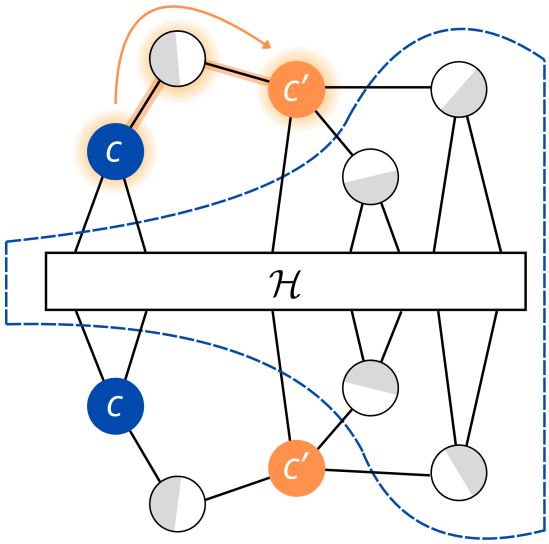

Figure 1.16: Moving the effective Hamiltonian from an optimization center c to another optimization center c'. Only the renormalized Hamiltonian terms defined on the links located on the path from c to c' have to be updated (orange shaded). [33]

Optimizer strategies

The typical strategies employed in the numerical optimization problem are the *single-tensor update* and the *double-tensor update*. The two methods differs in the number of optimization centers involved, in the former one tensor is used, while in the latter two. It is also possible to define also a *single tensor update with a subspace expansion*.

Choosing one scheme rather than another relies on the variational problem under study and on the available computational resources. The first two strategies, single- and double-tensor update, are a generalization of DMRG to arbitrary loop-free TN architectures and are related to the traditional single-center and double-center site DMRG respectively. On the other hand, the subspace expansion

scheme provides a way to exploit the improved convergence of the double-tensor update, while keeping the computational cost scaling of the single-tensor update.

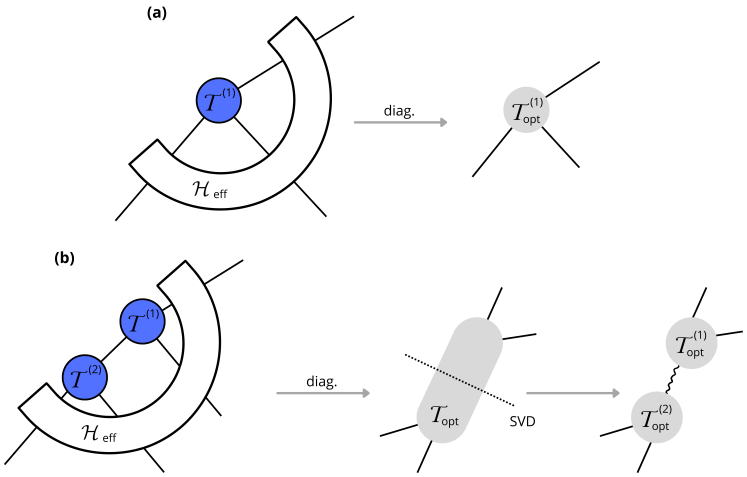

Figure 1.17: (a) Single-tensor update. The reduced diagonalization problem is formulated for one tensor. The optimized tensor is given by the eigenvector of H_{eff} with the smallest eigenvalue. (b) Double-tensor update. The reduced diagonalization problem is formulated for two tensors. The number of links of the optimized tensor is given by the number of links on which H_{eff} acts. The link resulting from the SVD can differ from the original link, if the SVD performs a non-trivial separation of the links of T_{opt}. [33]

In the single-tensor update, the optimization center is just one single tensor and thus it represents the most simply and computationally inexpensive strategy. However, when dealing with a Hamiltonian which exhibits a symmetry, the same symmetry will be present also in the effective Hamiltonian of the reduced diagonalization problem. Thus, the local optimization must preserve the link representations on the individual links of the optimization center, leading to the fact that the quantum numbers can not be handled variationally. For tensor networks made of tensors that do not share symmetries, the constraints are softer: in this case they are only metastable configurations which can be violated through numerical fluctuations, caused e.g. by finite precision floating point operations or by the fact that the environment (i.e. the tensors surrounding the optimization center) does not obey the symmetry. In order to solve this problem, the diagonalization may be formulated for more adjacent tensors, usually two, that have been previously contracted together. After the optimization of the compound tensor, the original tensor is restored through SVD-decompositions. The accompanying truncation of the state spaces associated with the compound's internal links provides the variational handle lacking for external links (Fig. 1.17). Since the number of links involved is larger, the multi-tensor strategy exhibits a higher computational cost.

To mitigate the high computational cost, one can exploit the single-tensor subspace expansion optimization strategy. This consists in temporarily enlarging the state space of one virtual link of the selected tensor, in order to allow for a variation of the symmetry representation on that link. Subspace expansion methods in single- and double- tensor updates may become an even more powerful technique when combined with perturbative approaches, e.g. for curing the shortcomings of the traditional single-center site DMRG.

Optimizer execution

The diagonalization routine of the effective Hamiltonian represents a fundamental part in the optimization loop, since the total computational cost heavily depends on this. Fortunately, thanks to the installed unitary gauge, the tensor network is easily contractible and the eigenvalue problem reduces from a generalized form to a standard one. Since the effective Hamiltonian is sparse and only the

ground state eigenstate is required, one can benefit from the use of iterative eigensolvers based on power methods, such as Lanczos/Arnoldi iteration. These algorithms require only the action of the effective Hamiltonian on the center node tensor, $H_{eff}\,|T^{[c]}\rangle$, and are finalized in returning only few eigenpairs. The computational cost of the different update schemes thus depends on the numerical effort for evaluating the action of the effective Hamiltonian on the tensor. In particular, it is a function of the number of application of the eigensolver, the number of terms of the effective Hamiltonian and the dimension of the virtual links.

1.1.3 Loopy tensor networks

In many scenarios, it is more convenient to build specific tensor network ansätze with cycles; these network geometries are commonly referred to as *loopy tensor networks*. Various topologies have been studied, including infinite-MPS (iMPS), Projected Entangled Pair States (PEPS) for simulating two-dimensional quantum many-body (QMB) systems, Multiscale Entanglement Renormalization Ansatz (MERA) for studying hierarchical scale-invariant systems, and branching MERA, among others. However, all these tensor networks present difficulties when implementing optimization algorithms, either due to less favorable scaling or because they are prone to numerical instabilities. Consequently, implementing these networks requires additional care and higher expertise.

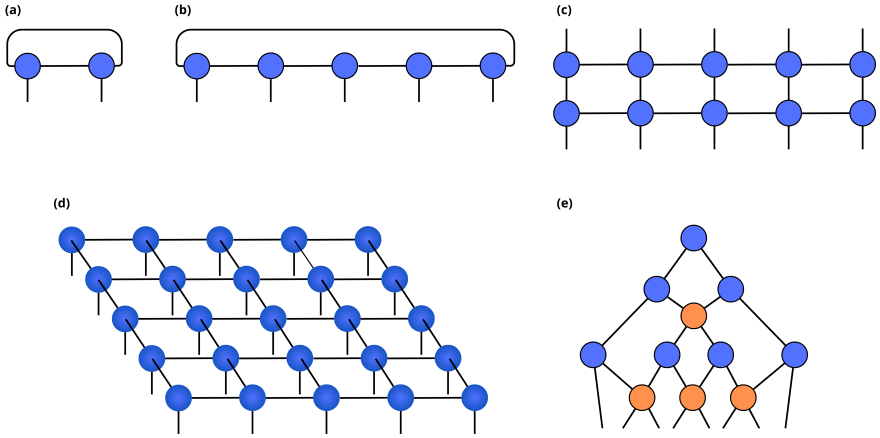

Figure 1.18: (a) Infinite MPS (iMPS). (b) MPS with periodic boundary conditions. (c) Locally purified tensor network. (d) Projected entangled pairs state (PEPS). (e) Multi-scale entanglement renormalization ansatz (MERA), composed by unitaries (orange tensors) and isometries (blue tensors). [36]

Loopy tensor networks can efficiently represent many-body quantum systems, with particular geometries and structures, even in dimensions higher than one, and can capture more entanglement content than simpler tensor network ansätze, which in contrast favor correlation clustering in accordance to the network geometry [28]. However, one of the main reasons loopy tensor networks are difficult to use is the absence of a unitary form for them. While loop-free tensor networks can always be transformed into their unitary gauge, many problems arise with the available linear algebra manipulations required to transform loopy tensor networks. Due to this issue, the exact contraction of these ansätze is not easy to compute: it is an NP-hard problem, as it scales exponentially with the number N of elementary constituents of the TN. Thus, when one needs to implement a ground state search algorithm, the diagonalization problem appears to be a generalized eigenvalue problem.

Focusing on PEPS ([5]), it is possible to introduce many features of these loopy ansätze. PEPS (Projected Entangled Pair States) are looped tensor networks built for representing 2D systems, thus capable of capturing a higher content of entanglement compared to loop-free geometries. PEPS are tensor networks that correspond to a 2D array of tensors, constructing a 2D lattice. Different types of lattice forms can be defined: square, honeycomb, triangular, etc.. Similar to the MPS (Matrix Product States) in loopless tensor networks, each PEPS tensor has two types of links: a physical link with dimension d_s and a virtual link with dimension D_η. PEPS exhibit some basic properties:

- 2D translational invariance and thermodynamic limit

  In general, one can arbitrarily impose all the tensors to be different, which lead to a not translationally invariant PEPS. However, one can impose this property and take the thermodynamic limit by choosing a fundamental unit cell, repeated all over the 2D lattice. As expected for higher-dimensional systems, translational invariance needs to be imposed in all the spatial directions of the lattice.

- PEPS are dense

  PEPS are able to represent any quantum state of the many-body Hilbert space by increasing the bond dimension. As for the loopless TN, the bond dimension needs to be exponentially large in order to cover the whole Hilbert space. However, for low-energy states and many 2D quantum systems, this dimension is expected to be small and finite.

- Area law

  PEPS also satisfies the area-law scaling of the entanglement entropy. The entanglement entropy of a block of boundary L of a PEPS with bond dimension D is $S(L) = O(LlogD)$.

- Polynomially decaying correlations

  PEPS can manage two-point correlation functions that decay polynomially with separation distance. This property is significant because polynomially decaying correlation functions, as opposed to exponentially decaying ones, are characteristic of critical points where the correlation length is infinite and the system exhibits scale invariance. Therefore, the class of PEPS is, in principle, suitable for describing both gapped phases and critical states of matter.

- NP-Hard exact contraction

  The exact calculation of the scalar product between two PEPS is an exponentially hard problem. This means that for two arbitrary PEPS of N sites, it takes a time $O(exp(N))$. It is possible to approximate these expectation values using numerical methods.

- No canonical form

  It is not possible to reduce the PEPS in canonical form, since it is not possible to choose an orthonormal basis for all the bond indices. This happens as long as a TN geometry possesses a loop. In fact, a loop in the TN means that one can not formally split the network into left and right bipartition by just cutting one index, thus a Schmidt decomposition between left and right is not defined. Approximate numerical methods have been implemented to reduce the non-critical PEPS in a quasi-canonical form in order to find ground states.

1.1.4 The unitary form

As introduced in the previous paragraphs, the possibility to put in unitary form a tensor network is fundamental. The unitary form tremendously reduces the computational time required to contract the TN. Contractions, in fact, are the building blocks of every algorithm which aims to compute physical quantities, such as ground state search algorithms. In the unitary form, one can exploit intensively the use of the isometries introduced in the gauging procedure, to effectively annihilate some tensor contractions between the TN state and its conjugate. This allows to recast the whole TN contraction to a contraction of a smaller subset of tensors (for norms) or a braket with some local operators (for expectation values). Loop-free TNs can be transformed in their unitary form, by installing specific gauges (e.g. unitary gauge). On the contrary, loopy TNs can not be transformed in this way and approximate numerical techniques needs to be used to extract the required quantities (approximated). This lack leads to another problem: in loops, spurious entanglement forms. Whenever a loop of tensors is present in a TN ansatz, the content of entanglement present between the tensors could be higher than expected. However, it is not possible to eliminate such false correlations, since the contractions

required to annihilate it are not available, due to the non possibility to put in a unitary form the tensor network.

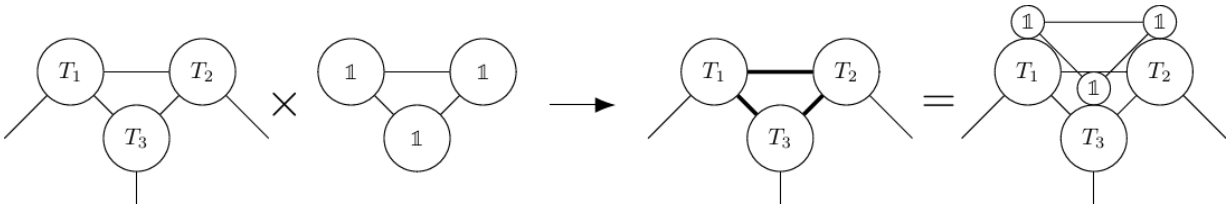

Figure 1.19: Example of spurious entanglement. Correlations between the tensors, in a larger TN, may be in a superposition with a false correlation generated by the interaction between the auxiliary spaces. Once this false entanglement is formed, one can not remove it.

This spurious entanglement forms between virtual link spaces and does not represent a physical correlation. It slows down the implemented algorithm by occupying more memory and requiring more operations than necessary. One possible solution is to enlarge the auxiliary link spaces. However, this approach wastes additional resources, as the virtual spaces now describe more useless information. Moreover, this operation further slows down the protocol because the contractions, which scale with the involved dimensions, require more computational time. Once a loop is formed and spurious entanglement appears, it is not possible to find the original state with "minimal" entanglement using standard linear algebra tools.

The aim of this thesis is to build a procedure to put in unitary form a loopy tensor network, by building a multi-linear algebra manipulation of the tensor network.

Chapter 2

Entanglement distribution decomposition

Tensor Networks ansatz states are a class of prominent numerical methods for variational simulation of many-body quantum systems. They faithfully, and efficiently, represent states of physical systems close to equilibrium, for example on lattices with finite-range interactions, and especially in low physical dimensions. Tensor Network geometries without loops display the most powerful and stable algorithms, but they struggle when attempting to reproduce the entanglement distribution of higher-dimensional states. Conversely, loopy tensor networks can accommodate entanglement to fully capture 2D and 3D interacting models, but algorithms are drawn back from a complexity in removing spurious correlations around the loops, which slows down the simulation without carrying meaningful physical information.

Spurious loop correlations could be in principle removed by starting from a tripartite entanglement-based decomposition of rank-3 tensors along a network loop. Specifically, the problem of efficiently decomposing a tripartite tensor into a small tensor network (3-6), with three tensors and six total links (three external, three internal) is addressed in this chapter. An efficient decomposition is such that each correlation between the three parties takes the shortest route along the (3-6) network. This objective translates, practically, into decomposing a three-leg tensor into a small network of three tensors (multilinear algebra decomposition). The new network has to reproduce, either exactly or approximately, the starting tensor but at the same time employing the smallest possible dimensions of the internal, newly created virtual correlation spaces. Carrying out this task involves addressing a complex optimization problem, which might be tackled with direct search strategies as well as machine learning strategies. A numerical algorithm to acquire such efficient tripartite decomposition, which will play the role of a key component when designing and running algorithms for loopy tensor networks, is explained in this chapter.

Once the optimized decomposition is available and functional (and properly trained, in the case of an AI) the final step will be to explore strategies to employ it in loopy tensor network methods, focusing on techniques to remove spurious virtual correlations in loopy tensor network ansatz states.

2.1 Framework

As introduced in the previous chapter, sec. 1.1.3, loopy tensor networks can not be transformed into the unitary via single-tensor standard linear algebra operations form due to the presence of cycles. This hindrance leads to difficulties in removing spurious entanglement around the network loops, because the required contractions, through the isometrization of the tensors, are not available with the existing tools. Thus, all the algorithms designed for loopy TN require an intensive use of the resources and a high computational time and comparatively larger approximation errors.

In order to build a procedure for removing the spurious correlations and transforming in unitary form a loopy tensor network, several steps need to be accomplished. The first objective is to isometrize

all the tensors in a cycle with respect to one edge, the center node. A fundamental part of this protocol is understanding how correlation between the three parties takes the shortest route along the (3-6) network. This problem is addressed by the multi-linear algebra algorithm developed in the next section. Then, the procedure is iterated all over the tensor network, isometrizing all the nodes with respect to the selected isometrization center.

The steps required to isometrize a loop are now introduced. Start from a cycle of four tensors inside a larger loopy tensor network. Define arbitrarly the upper-left tensor T_1 as the center of the isometrization, the other tensors need to be isometrized in its direction.

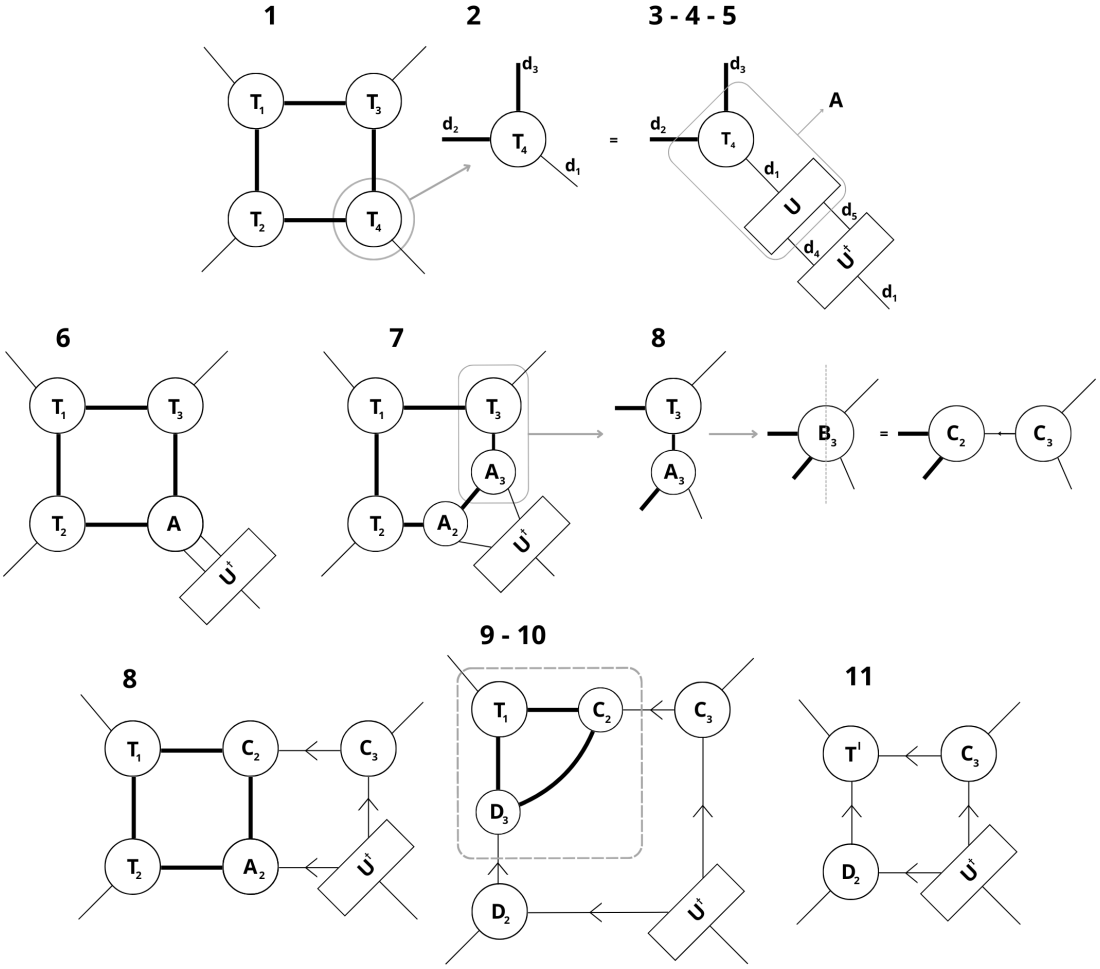

Figure 2.1: Procedure required to isometrize a loopy tensor network.

1. Select the farther loop from the target unitary center

   In general, a cycle contains spurious entanglement, which require a larger virtual link dimension to accomodate it. Unfortunately, it is not possible to know if false correlations are truly present. Thus, without the following prescription, the link dimension can not be lowered.

2. Extract the tensor in the outermost level with respect to the isometrization center in the loop

   Select the tensor, T_4, in the outermost level with respect to the one we want the isometrization. This is a rank-3 tensor with dimensions d_1, d_2, d_3. We consider the leg of dimension d_1 to be the one outside the selected loop.

3. Insert a unitary identity

   Along the link not inside the loop, insert a unitary identity $UU^\dagger = \mathbb{1}$. The unitary matrix has thus two legs of dimension d_1.

4. Split the leg between the unitary matrices

Through link splitting (sec. 1.1.1), divide the leg into two factor legs of dimensions d_4 and d_5. The product of the new leg dimensions needs to be equal to the original divided leg dimension, $d_4 \cdot d_5 = d_1$. The optimal splitting is an operation that needs to be tuned, the algorithm in the next section will provide a guide on how to do it.

5. Contract the tensor T_4 and the unitary matrix U

   By contracting the tensor and the unitary matrix, we obtain a new rank-4 tensor A of dimensions d_2, d_3, d_4, d_5. The adjoint unitary matrix replaces the original tensor T_4 in the edge.

6. Apply the algorithm on tensor A and unitary matrix U

   Use the multi-linear algebra decomposition explained in the next section to optimize the unitary matrix U and perform the entanglement distribution decomposition. Thus, correctly distribute the correlations between the other tensors in the loop. Then, decompose the tensor A into two tensors A_2, A_3. The original edge composing of a single tensor is now a triangle-like network composed of three tensors.

7. Contract the tensor A_3 into its respective neighboring tensor T_3

   The contraction involves two rank-3 tensors T_3 and A_3. The result is a rank-4 tensor B_3.

8. Decompose the tensor B_3 into two factor tensors C_2, C_3

   After the decomposition, we obtain a tensor C_2 connected with A_2 and a tensor C_3 replacing the tensor T_3 in the edge isometrized toward T_1. In total, we have six tensors in the loop.

9. Perform step 7 and 8 for the tensor T_2, obtaining two factor tensors D_2, D_3

   In this way, we reproduce the same geometry obtained for the tensor T_3. Thus, we have in the edge of T_4 the adjoint unitary matrix $U^\dagger$, in the edge T_3 the factor tensor C_3 isometrized toward the edge T_1 and in the edge T_2 the factor tensor D_2 again isometrized toward the edge T_1.

10. Contract the three tensor in the neighbor of the tensor T_1

    Tensor T_1 in the edge and the two factor tensors C_2, D_3 arising from the two decomposition from edge T_3 and T_2 are contracted together to replace the tensor T_1. The output is a rank-3 tensor T'.

11. The loop is now isometrized from the edge T_4 toward the edge T_1

    Thanks to the multi-linear algebra algorithm, the decompositions and the contractions, the loop is now isometrized and can be efficiently contracted with its conjugated. The final loop presents in the edges the tensors T', D_2, C_3, $U^\dagger$.

12. Move to the next loop closer to the unitary center

    Iterate the procedure, moving closer to the unitary center, until all the tensor network is isometrized towards the center node.

After the tensor network is isometrized, efficient contractions can be leveraged (Fig. 2.3). Indeed, focusing on a single loop, we can note that thanks to the reverse isometrizations installed, all the contractions are annihilated, except the one of the isometrization center. The term *reverse isometrization* refers to the idea that this kind of isometrization points from the side of the tensor with one leg to the one with two legs, thus leading to two identities. This is in fact the contrary of the isometrization prescription induced by the unitary gauge in loop-free tensor networks.

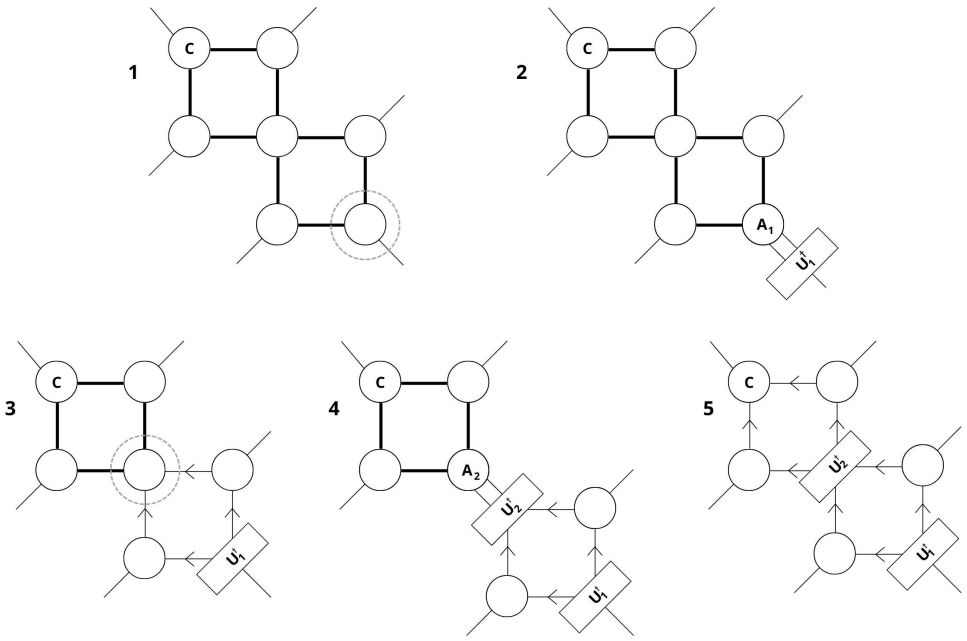

Figure 2.2: Practical visualization of the procedure applied on two loops. The target unitary center is depicted as the node c. Starting from the outermost level (the farther node in the farther loop), move to the closer tensors and closer loop.

In reference to the figure 2.3, by contracting a loop from a larger TN with its adjoint, we can see that the contraction over the edge T_4 is annihilated, i.e. it is an identity, as well as the contractions over the edges T_2 and T_3. In the end, only the contraction over the isometrization center T_1 remains to be executed: two internal links arising from the loop links and one external link, that was connected with the rest of the network, need to be contracted with their adjoint.

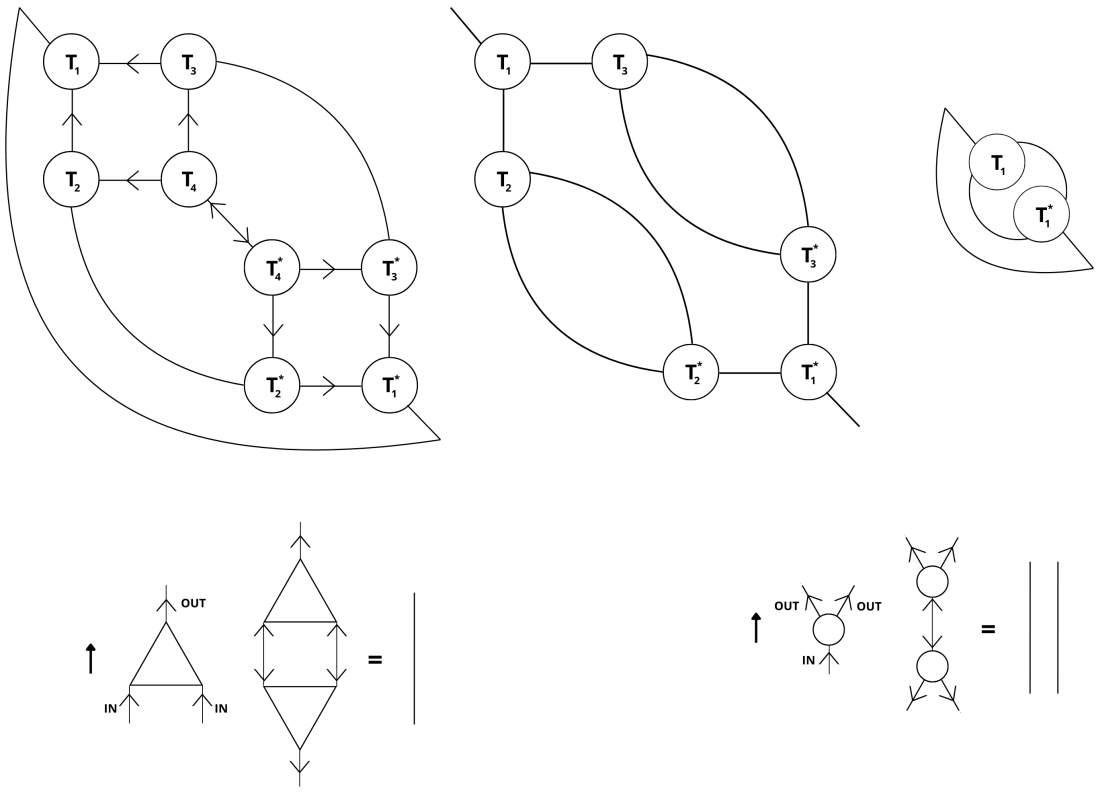

Figure 2.3: Top: efficient contraction of a loop with its adjoint. Bottom: (left) isometrization for loop-less tensor networks and (right) reverse isometrization for loopy tensor networks.

2.2 Entanglement distribution as an optimization problem

In this section, the multi-linear algebra entanglement decomposition algorithm is introduced as an optimization problem. This protocol plays a predominant role in the procedure of isometrization of loopy tensor networks, as it can be seen in point 6. Two strategies are analyzed: direct search method and AI neural network.

The entanglement distribution decomposition takes as input a rank-3 tensor with legs of dimensions d_1, d_2, d_3. This tensor represents a quantum state and thus it possesses a precise underlying structure, which unfortunately is not know a priori. Such a structure could tell us how the correlation between the parties in the loop is distributed. In order to better understand this passage, the five most probable structure, which represent the benchmark (training instances) for the performances of the algorithm, are reported here:

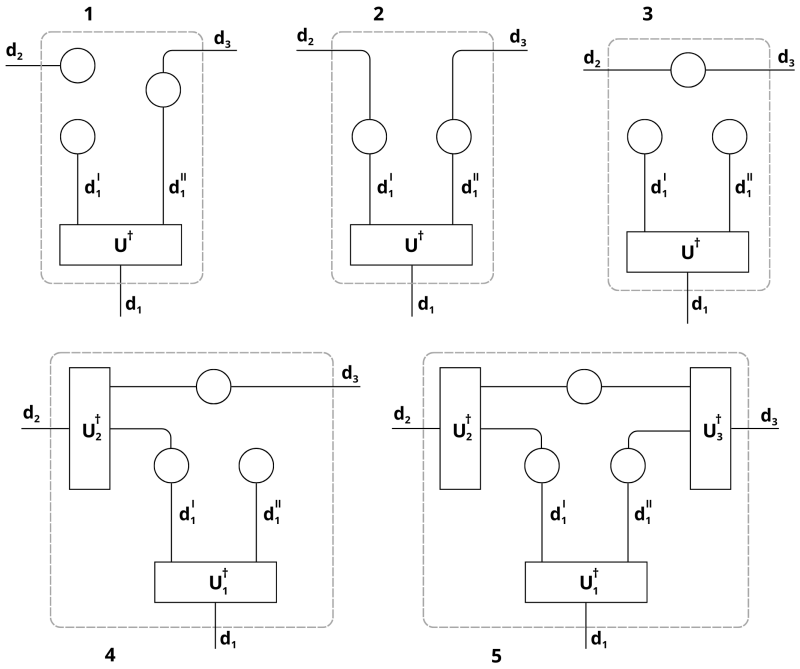

Figure 2.4: Five benchmark (training) cases to test the algorithm performances. The rank-3 tensors possess an underlying structure composed by single particle and many-body quantum state.

In reference to the figure 2.4, the single-leg nodes represent random single particle wavefunctions, encoded as tensors. The double-leg nodes represent instead many-body quantum systems, specifically random states or generalized Bell Pair states, encoded again as tensors. The dimensions of the physical legs are not fixed and need to be selected to simulate the investigated quantum system, while the auxiliary leg dimensions are fixed to one. The entanglement between the bipartitions of the system are thus represented by the many-body wavefunctions, which host the correlations between the three legs of the tensor.

A support unitary matrix is employed to reproduce the rank-3 tensor starting from the underlying structure, thus fusing the legs d_1', d_1'' into the tensor leg of dimension d_1. The same procedure is applied, if required, to the other underlying legs composing d_2 and d_3. In fact, in the loopy tensor network, only rank-3 tensors are present.
As it can be seen from these geometries, by performing a SVD-decomposition, which vertically cut the tensor in the middle dividing it in a left and right partition, one could in principle understand if there is correlation between these bipartitions. Indeed, with the singular values λ_s arising from the SVD, one can evaluate the Von-Neumann entropy, which represents an entanglement measure between the left and right sub-systems:

$$S = -Tr(\lambda_s^2 log \lambda_s^2) \tag{2.1}$$

Furthermore, the same operation could be also performed by cutting the tensor horizontally, thus diving the tensor in an upper and lower partition and measuring the correlation between these two reduced systems. However, these operations can not be performed since the tensors in the loop are rank-3. Consequently, in order to transform the node into a rank-4 tensor, a unitary identity, consisting of a unitary matrix and its adjoint, is introduced in the link d_1 external to the cycle. The leg between the unitary matrices is then split into two factor legs d_4, d_5. In this way, the original rank-3 tensor can be contracted with the neighbor unitary matrix and it becomes a rank-4 tensor, i.e. with four legs. An SVD-decomposition can be accomplished, by dividing the new tensor in a left (upper) and right (lower) bipartition, and the singular values extracted. The Von-Neumann entropy can be evaluated.

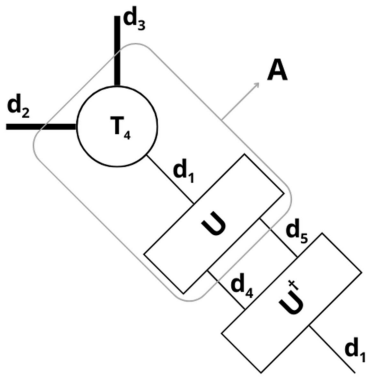

Figure 2.5: Rank-3 to rank-4 transformation through the insertion of a unitary identity $UU^\dagger = \mathbb{1}$. The adjoint matrix $U^\dagger$ does not enter the algorithm, it will replace the original rank-3 tensor in the loop.

The crucial part of the algorithm is that the unitary matrix needs to be properly tuned in order to return the minimal Von-Neumann entropy. Furthermore, the splitting of the leg d_1 into the legs d_4, d_5 also needs to be optimal, coinciding with the dimensions of the underlying structure or just reproducing the minimal entropy. These tasks are the core of the optimization algorithm developed in this thesis.

The optimization task can be defined as follows:

Given a rank-3 tensor T with dimensions $d_1 \times d_2 \times d_3$, and trial dimensions d_4 and d_5 constrained to $d_1 = d_4 \cdot d_5$, find the unitary matrix U such that the resulting tensor $A = TU$ minimizes the entropy S between the left and right partitions.

$$\textbf{Given:} \quad \text{Rank-3 tensor } T \in \mathbb{C}^{d_1 \times d_2 \times d_3} \quad \text{and dimensions } d_4, d_5 \tag{2.2}$$

$$\textbf{Optimize:} \quad \text{Unitary matrix } U \in \mathbb{C}^{d_1 \times d_1} \quad \text{such that } A = TU \tag{2.3}$$

$$\textbf{Minimize:} \quad S = -\mathrm{Tr}(\lambda_s^2 \log \lambda_s^2) \quad \text{with } \lambda_s \text{ singular values from SVD-decomposing } A \tag{2.4}$$

$$\textbf{Subject to:} \quad d_1 = d_4 \cdot d_5 \tag{2.5}$$

2.2.1 Direct search strategy

Several direct search methods are employed and tested to build the most efficient optimization algorithm. The optimization task is a minimization problem: the objective function to be minimized is the Von-Neumann entropy. The hyper-parameters, which need to be tuned, are the unitary matrix entries and the optimal splitting of the d_1 leg.
The following optimization algorithms are investigated.

- Nelder-Mead

The Nelder-Mead method [37], also referred to as the simplex method, is a non-linear optimization algorithm used for functions defined over an N-dimensional domain. This method is distinct from gradient-based algorithms, as it does not require derivatives. Instead, it relies on the concept of a simplex, a polytope with $N + 1$ vertices in an N-dimensional space, such as a line segment in a line, a triangle in a plane, a tetrahedron in space, etc.

Designed to find the local optimum of a function with N variables, assuming the function is smooth and unimodal, the Nelder-Mead method works by extrapolating the objective function's behavior at the simplex vertices. The algorithm replaces the worst-performing vertex with a new point, often the centroid of the remaining N vertices. If this new point improves the function value, the algorithm moves in that direction; otherwise, it explores other points that yield better evaluations.

The Nelder-Mead method employs various operations—reflection, expansion, contraction, and reduction—to iteratively refine its search, adapting dynamically to guide the optimization process towards convergence.

- Genetic algorithm

In the fields of computer science and operations research, Genetic Algorithms (GAs) [38] are a subset of metaheuristics inspired by natural selection principles, belonging to the broader category of Evolutionary Algorithms (EAs). These algorithms are designed to produce high-quality solutions for optimization and search problems by using biologically inspired operators like mutation, crossover, and selection.

In a genetic algorithm, a population of candidate solutions, also known as individuals, organisms, or phenotypes, evolves towards better solutions to an optimization problem. Each candidate solution has a set of properties, referred to as chromosomes or genotypes, which can be mutated and altered. While solutions are traditionally represented as binary strings of 0s and 1s, alternative encoding methods can also be employed.

The evolutionary process typically starts with a randomly generated population of individuals and progresses iteratively through generations. In each generation, the fitness of every individual is assessed based on the value of the objective function corresponding to the optimization problem. Individuals are then stochastically selected from the current population, and their genomes are modified through recombination and random mutation to create a new generation. This new generation serves as the basis for the next iteration of the algorithm. The algorithm generally terminates either when a predefined maximum number of generations is reached or when a satisfactory fitness level is achieved within the population

- Particle swarm optimization

In computational science, Particle Swarm Optimization (PSO) is a technique used to optimize problems by iteratively improving a candidate solution based on a given measure of quality. This method employs a population of candidate solutions, termed particles, which navigate the search space according to mathematical formulas based on their positions and velocities. Each particle's movement is influenced by its personal best-known position and is also guided towards the global best-known positions in the search space, which are updated as superior positions are found by other particles. This collaborative process aims to direct the swarm towards optimal solutions.

PSO is considered a metaheuristic because it makes minimal assumptions about the problem being optimized and can explore vast spaces of candidate solutions. Unlike classical optimization methods, such as gradient descent and quasi-Newton methods, PSO does not require the optimization problem to be differentiable, as it does not rely on the gradient of the problem. However, similar to other metaheuristics, PSO does not guarantee that an optimal solution will be found.

- Automatic differentiation

In mathematics and computer algebra, automatic differentiation (also referred to as auto-differentiation, autodiff, algorithmic differentiation, or computational differentiation) encompasses a set of techniques for calculating the partial derivatives of functions defined by computer programs. This approach takes advantage of the fact that all computer calculations, regardless of their complexity, are composed of a sequence of elementary arithmetic operations (such as addition, subtraction, multiplication, and division) and elementary functions (such as exponential, logarithmic, and trigonometric functions). By systematically applying the chain rule to these operations, automatic differentiation can compute partial derivatives of any order with high accuracy, maintaining working precision and requiring a computational effort that is only marginally greater than that needed to evaluate the original function.

Automatic differentiation is fundamentally different from both symbolic differentiation and numerical differentiation. Symbolic differentiation often encounters difficulties when attempting to convert a computer program into a single mathematical expression, leading to inefficient code. Numerical differentiation, which relies on finite differences, is susceptible to round-off errors and cancellation problems during discretization. Both approaches become increasingly problematic when computing higher-order derivatives due to escalating complexity and error rates, and they are inefficient for calculating partial derivatives with respect to many inputs— a critical requirement for gradient-based optimization algorithms. Automatic differentiation effectively addresses these challenges.

This technique is particularly important in the field of machine learning, where it enables the efficient implementation of backpropagation in neural networks without the need for manually derived derivatives.

2.2.2 Implementation

Among all the optimization algorithms analyzed, only the automatic differentiation procedure successfully finds an accurate solution in a short computational time. While the Nelder-Mead algorithm can locate the required minimum, it is highly inefficient, resulting in large computational time. The Genetic algorithm, on the other hand, struggles to find the minimum, often getting stuck in local minima and failing to achieve the necessary accuracy. The Particle Swarm Optimization (PSO) algorithm is both fast and accurate, making it a strong candidate for capturing the minimum of the Von Neumann entropy. However, automatic differentiation, implemented via backpropagation, achieves an unparalleled level of speed and precision, rendering even the PSO protocol inefficient by comparison. Consequently, the autodiff algorithm emerges as the ideal optimizer for the entanglement decomposition algorithm.

The performances of the algorithm are evaluated on test cases, before running it into a real loopy tensor network. Consider the five benchmark underlying structures (Fig. 2.4). The dimensions of the physical underlying legs are fixed to vary in the range $[2, 16]$. As introduced in the previous subsection, in order to transform the quantum state into a rank-3 tensor, a unitary matrix (its adjoint) is employed, fusing the underlying legs into one (e.g. d_1', d_1'' into d_1). Then a random unitary matrix is used to correctly distributing the correlations between the subsystems by minimizing the Von-Neumann entropy. The objective is tuning the random unitary matrix in order to reproduce, either exactly or approximately, the starting tensor (i.e. realizing $UU^\dagger = \mathbb{1}$) and returning in output a minimal Von-Neumann entropy.

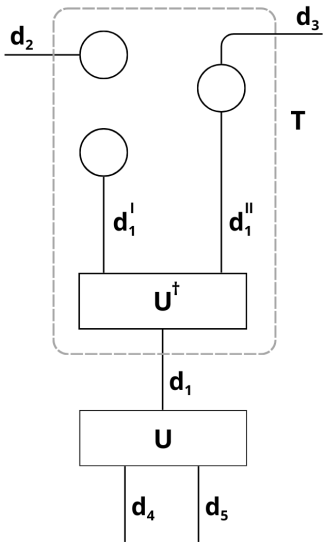

Figure 2.6: Starting from the underlying geometry of the quantum system, generate the rank-3 tensor, as the ones present in a loop of a tensor network, by contracting the structure with the $U^{\dagger}$. In this way, a semi-random rank-3 tensor is produced and the optimal value of the entropy is known. Consequently, randomly generate a unitary matrix U and contract it over the link d_1. Split the d_1 leg into factor legs d_4, d_5. Optimize the unitary matrix U in order to produce a minimal Von-Neumann entropy between left and right bipartition.

The algorithm thus starts by generating the random unitary matrix of dimension $d_1 \times d_1$, respecting the Haar measure [39], and contracting it over the link d_1 of the rank-3 tensor T (reference to Fig. 2.6).

The protocol is composed by three main parts. Since the underlying structure of the rank-3 tensor is in principle defined, one could exploit simple entropy evaluations to guess the correct geometry and thus perform optimally the optimization tasks: unitary matrix entries tuning and correct leg d_1 splitting individuation.

The tuning procedure, as introduced before, is an automatic differentiation algorithm implemented via backpropagation using the autograd library of Pytorch [40]. The algorithm takes as input a rank-3 tensor, contracts over the link of dimension d_1 a randomly generated unitary matrix $d_1 \times d_1$, splits the remaining d_1 leg into two factor links d_4, d_5, thus reshaping the final rank-4 tensor into the form (d_2, d_4) vs (d_3, d_5), performs the SVD-decomposition and computes the Von-Neumann entropy between left and right bipartition. The backpropagation enters the procedure by automatically constructing a gradient-like tensor for the unitary matrix. At each iteration, the unitary matrix is updated through its numerical gradient, consequently the entries of the matrix change at each step. Within the new unitary, the contraction over the link d_1 of the rank-3 tensor is again performed, the SVD-decomposition is accomplished, dividing the rank-4 tensor into two partition, and the Von-Neumann entropy is computed, aiming to lower at each iteration its value to the minimum. The algorithm halts after completing the maximum number of allowed iterations.

The three passage of the algorithm are now introduced:

1. Vertical SVD-decomposition

   The first step is to evaluate the Von-Neumann entropy between the left and right bipartition of the system (S_{vert}), by vertically cutting the tensor through an SVD-decomposition. Starting from the leg dimension d_1, the Von-Neumann entropy is evaluated for all the possible splitting $\{[d_4, d_5] : d_1 = d_4 \cdot d_5\}$. This passage does not require a high computational time, given the dimension in use.

2. Horizontal SVD-decomposition

   The second step is to evaluate the Von-Neumann entropy between the upper and lower bipartition

of the system (S_{horiz}), by horizontally cutting the tensor through an SVD-decomposition. It is not required to test all the possible splitting of the leg d_1, it is sufficient to test one.

3. Final vertical SVD-decomposition

   The third step relies on analyzing the vertical and horizontal entropies, thus identifying the underlying structure and selecting the optimal splitting. Once the optimal splitting is found, the autodifferentiation algorithm is applied, tuning the unitary matrix entries and returning the minimal entropy.

   - Completely separable case

     If both the splittings $d_1 \to d_4, d_5$ corresponding to $[d_4 = d_1, 1]$, $[1, d_5 = d_1]$, yield zero vertical entropy, then the tensor is completely separable and thus no optimization is required, since every unitary matrix with these splitting realizes the minimal entropy, equal to zero in this case.

   - Case 1

     If the vertical entropy related to the splitting $[d_4 = d_1, 1]$ is zero, then the optimal splitting returned is the one with the dimension $d_4 = e^{S_{horiz}}$. If instead the vertical entropy related to the splitting $[1, d_5 = d_1]$ is zero, then the optimal splitting returned is the one with the dimension $d_5 = e^{S_{horiz}}$.

   - Case 3

     If the entropies related to the splittings $[d_4 = d_1, 1]$, $[1, d_5 = d_1]$ are equal, then every splitting realizes the minimal entropy. Furthermore, the horizontal entropy is always zero, for every possible splitting.

   - Case 2-4-5

     In these cases, do not consider the splittings $[d_4 = d_1, 1]$, $[1, d_5 = d_1]$, since they belong to the previous possibilities. For all the other splittings, start the tuning algorithm and iterate for a small number of steps. For the case 4, a pattern can be identified: a series of equal entropies following or followed by higher values of entropies appears. Thus, the optimal splitting to be selected is the first or last one of the series, thus the one closer to the other higher values of entropy. For the other cases, the optimal splitting is the one which realizes the smallest entropy in the reduced tuning procedure (small number of iterations).

   After the optimal splitting is found, the tuning algorithm is applied for an arbitrary required number of iteration. The entanglement decomposition unitary matrix is computed.

2.2.3 AI strategy

In order to automate the process of finding the optimal splitting, an AI strategy is analyzed. The goal is to build a neural network (NN) to find the correct splitting of the tensor leg of dimension d_1 into two factor legs of dimensions d_4 and d_5. The loss function to be minimized is the Von Neumann entropy, so the same steps introduced before for tuning the unitary matrix are performed. Backpropagation is used to construct the numerical gradient of the unitary matrix. Various architectures are tested and evaluated to find the correct splitting. The NN is built using the PyTorch [40] library.

The first design tested is a Convolutional Neural Network (CNN). The idea is to learn correlations by inspecting the tensors as images. Tensors are input into multidimensional convolutional layers, followed by pooling layers. The output of the convolutional layers becomes the input of linear layers, ultimately returning one number representing the dimension d_4 of the correct splitting.

The second architecture investigated is a Fully-Connected (FC) neural network, composed of many fully-connected layers. This design requires the input tensors to be flattened. Again, the network outputs one number representing the dimension d_4 of the correct splitting.

Several challenges arise with the AI approach. Firstly, all input tensors need to be of the same shape. However, the optimization task inherently requires dealing with tensors of different shapes, as the quantum state represented has physical legs of different dimensions. Consequently, a padding stage is employed to standardize all tensor dimensions to the maximum dimension present in the dataset. This results in inefficiencies, as even low-dimensional tensors require higher memory content when padded. The padding also disrupts the image-like correlations learnable by the CNN. Similarly, the FC neural network is disadvantaged by the increased memory requirements, necessitating smaller tensors. To address this, an autoencoder is employed to compress the tensors. However, the padding stage biases the compressible information, making the autoencoder ineffective at reconstructing the original tensor and reducing its size.

Nevertheless, the fully-connected neural network represents the most promising architecture, as it could potentially learn the features once the input tensor sizes are fixed to a manageable size. This neural network is composed of multiple fc-layers. The first layer has an input dimension corresponding to the size of the tensor after the padding stage. Subsequent layers progressively increase in dimension to enhance the network's capacity for learning complicated correlations. Finally, the dimensions are reduced through the remaining layers, ultimately producing a single output value. The network operates on real numbers, but the input tensors are complex. To address this, the real and imaginary components are separated and processed through distinct sub-networks, which are then recombined before the final layer to produce the output.

The training and test datasets are composed of several complex-valued tensors, representing all five possible benchmark cases with varying shapes. Training such a neural network is challenging, so the underlying dimensions d_2' and d_3' are fixed at 2 and 5, respectively, while d_1' varies in the range [2, 6]. In this way, the maximum rank-3 tensor (after employing unitaries matrix to contract the underlying geometry of the quantum system) shape is $[d_1 = 30, d_2 = 12, d_3 = 10]$. The dataset includes also the original tensor shape before padding, the correct splitting, and the entropy of a vertical SVD with the optimal splitting. The loss function is composed of different penalties to better force the algorithm to learn the optimal splitting. The original dimension d_1 is known and used as a label in the algorithm. With the NN's output d_4, the dimension d_5 can be determined. The first penalty ensures $d_4 \cdot d_5$ matches d_1, and the second penalty minimizes the Von Neumann entropy, assuming infinite value if the splitting is incorrect ($d_4 \cdot d_5 \neq d_1$).

Unfortunately, the AI approach does not succeed in producing the required splitting. A larger network and an enhanced tensor dataset might solve the problem.

Given the difficulty of finding integer numbers for a neural network, an alternative approach could be producing the optimized unitary matrix alongside the correct splitting. However, training such an NN requires considerable time, representing a future research direction.

Chapter 3

Results

The entanglement decomposition algorithm is evaluated across five benchmark cases. These cases, as introduced in the previous chapter, possess an underlying geometry that allows for an understanding of how the correlation in the tensor is distributed. In principle, the physical leg dimensions and the auxiliary leg dimensions could assume any value, consistent with the physics underlying the quantum system. For this evaluation, the physical links are varied in the range $[2, 16]$, while the auxiliary ones are fixed to one. This choice of constraints is necessary to keep the computational time manageable, as higher dimensions would require more computational power, typically available only in high-performance computing infrastructures.

The AI-based approach fails to complete the optimization task successfully. Achieving the required accuracy and convergence speed would necessitate a more detailed neural network and a more powerful computing environment. Conversely, the direct search approach successfully reaches the intended goal.

3.1 Evaluation metrics

The metrics for evaluating and monitoring the objectives are the accuracy of the calculated Von Neumann entropy between the left and right partitions of the tensor and the computational time required. The correct entropy is calculated based on the known underlying structure, with accuracy measured as the discrepancy between the extracted value and the real one. The time of convergence monitors how the algorithm scales with the dimensions under consideration. Among these metrics, a plot representing entropy minimization as a function of the steps is provided, alongside bar plots representing the entropies evaluated for all possible splittings. This visualization allows us to understand how the correct splitting is selected, as described in the previous chapter.

3.2 Iteration parameters

The number of iterations required for reduced optimization to find the optimal splitting in the 2-4-5 cases needs careful selection. Too many iterations might result in an ill-conditioned matrix due to many singular values being equal, causing the algorithm to fail. On the other hand, too few iterations might prevent the correct identification of the division. Once this parameter is defined, the final number of iterations used for tuning the unitary matrix entries within the correct splitting is selected. A higher number of iterations translates to higher accuracy but also increases computational time, while fewer iterations have the opposite effect. Both the reduced and final number of iterations need to be chosen based on the specific dimensions in use. Here, the final number of iterations is tuned for each test case and the reduced number of iterations is fixed to about one third of the final iteration number.

3.3 Learning rate dynamics

As observed in the entropy minimization plots on a logarithmic scale, the learning rate plays a crucial role in the model's performance and requires careful tuning. This study introduces a dynamic learning rate approach that adjusts based on the specific tensor being analyzed. In all cases, the learning rate is initially set to 10 and undergoes exponential decay. The decay rate is fixed at 0.96, with the decay step being a percentage of the total (reduced) number of iterations, tailored to the tensor in question. The learning rate follows the formula:

$$lr = initial\_lr \cdot decay\_rate^{\frac{iteration\_step}{decay\_step}} \tag{3.1}$$

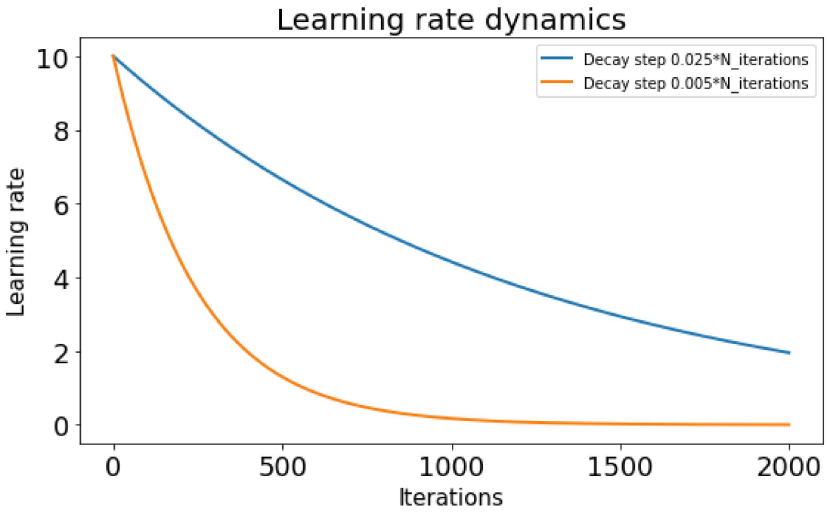

Figure 3.1: Learning rate dynamics for two different decay steps.

This approach allows the algorithm to be faster in the initial phase, where entropy decreases rapidly, while slowing down as it approaches a minimum to achieve higher accuracy. The learning rate can be further tuned for specific cases to improve performance.

3.4 Algorithm's results

Various underlying physical leg dimensions, $(d'_1,\ d'_2,\ d'_3 = d''_1)$, are tested. The dimensions reported here are chosen to allow for many possible splittings, providing a robust evaluation of the algorithm's performance. The leg d_1, originating from the underlying legs d'_1, d'_3, is selected to have many divisors. In all the results, the algorithm successfully retrieves the correct splitting. In the reported results, numerical values of the order 10^{-16} are considered as zeros, since they are close to the machine precision.

- Case 1

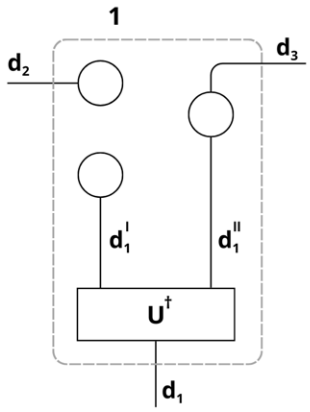

Figure 3.2: Benchmark case 1

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 3$	0.0	0.0	1.72	1000

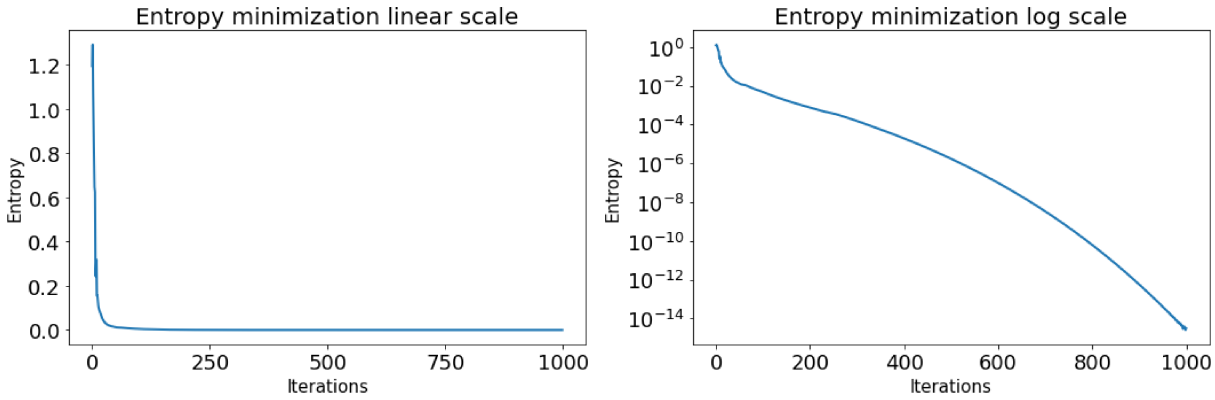

Figure 3.3: Entropy minimization as a function of the iterations

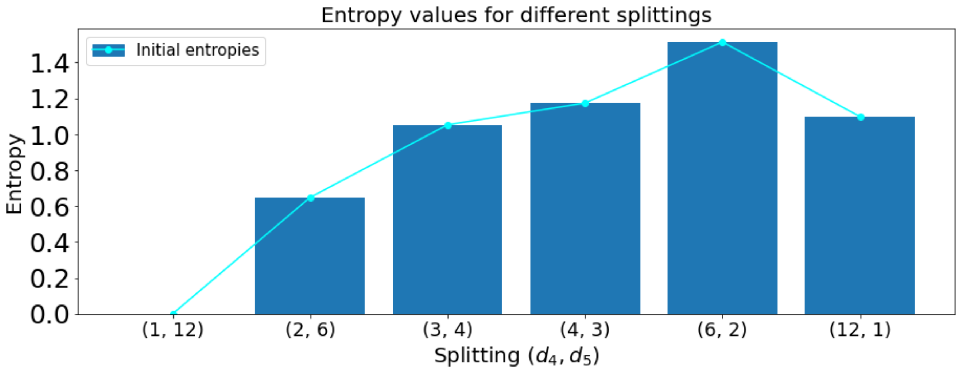

Figure 3.4: Bar plot with entropy evaluations for every splitting

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 6$	0.0	0.0	6.65	2000

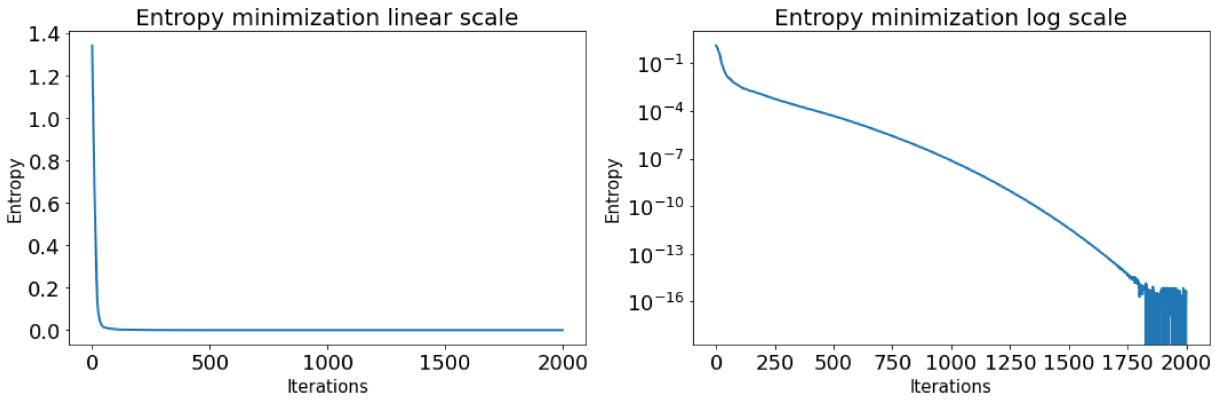

Figure 3.5: Entropy minimization as a function of the iterations

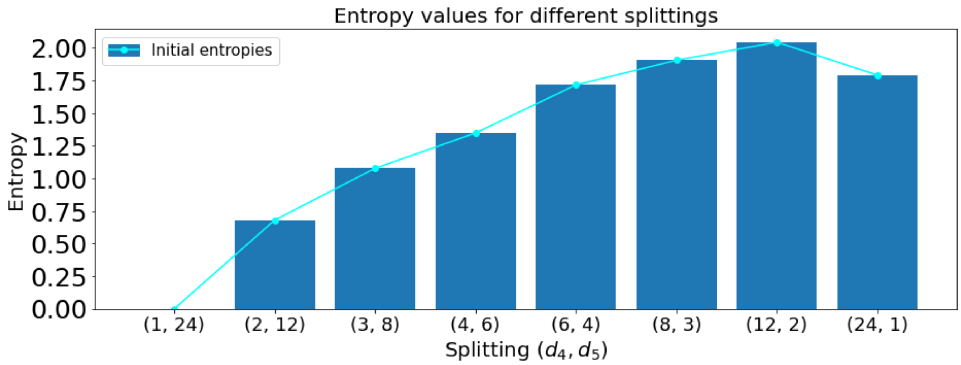

Figure 3.6: Bar plot with entropy evaluations for every splitting

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 8\ d_2' = 9\ d_3' = 9$	0.0	0.0	19.24	2000

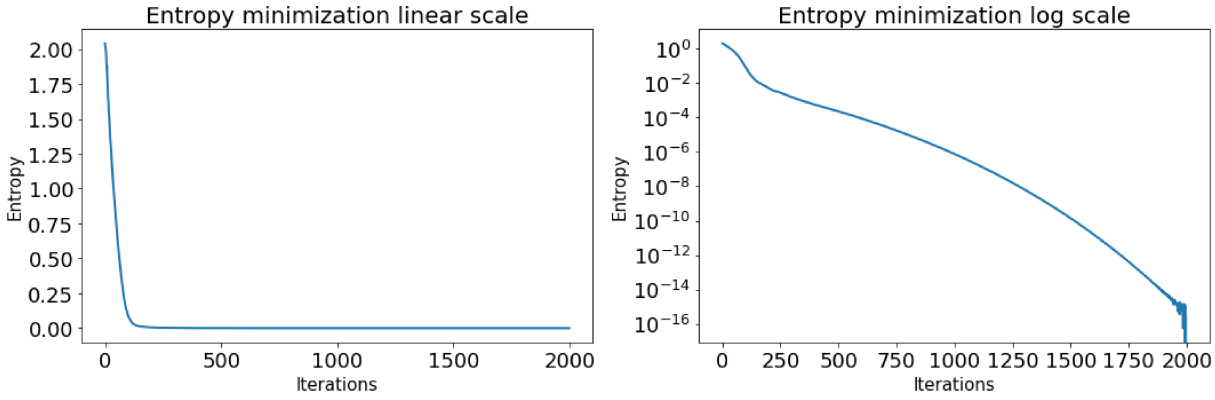

Figure 3.7: Entropy minimization as a function of the iterations

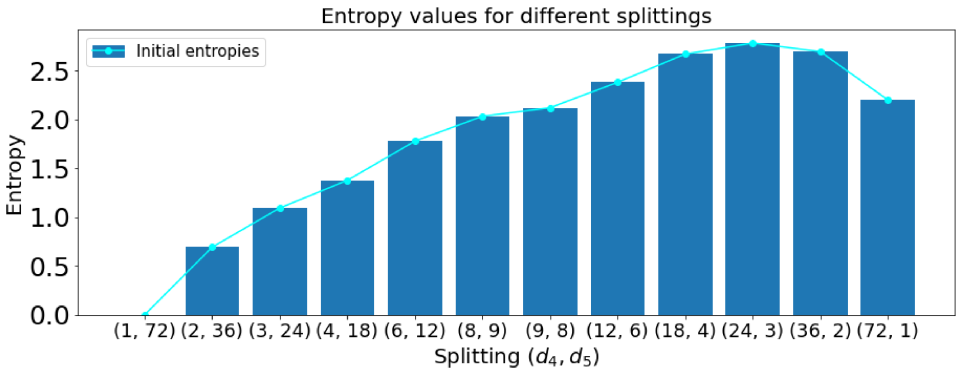

Figure 3.8: Bar plot with entropy evaluations for every splitting

- Case 2

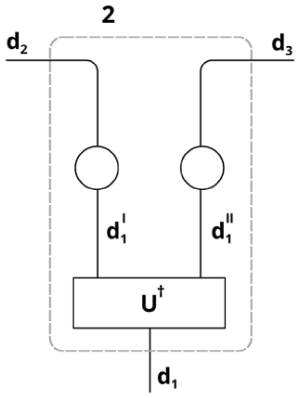

Figure 3.9: Benchmark case 2

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4$ $d_2' = 9$ $d_3' = 3$	0.0	0.0	3.84	(350) 1000

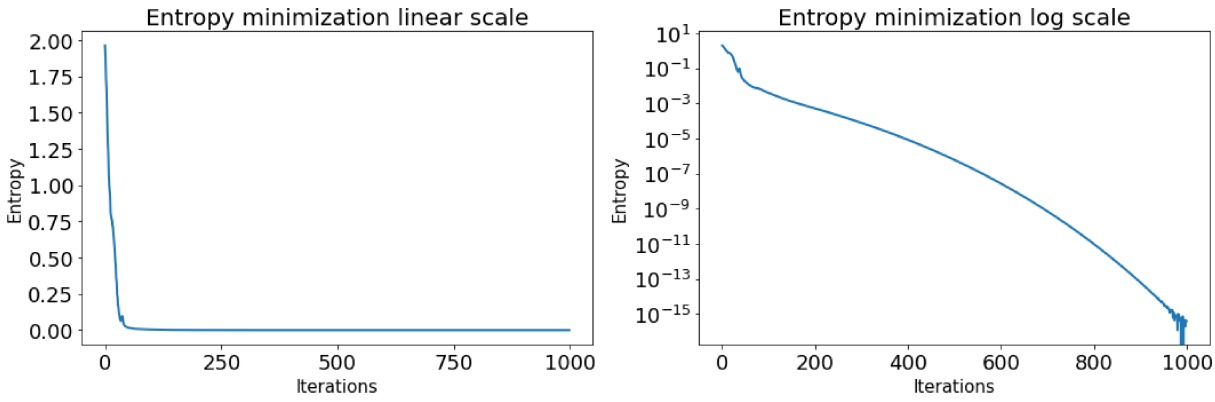

Figure 3.10: Entropy minimization as a function of the iterations

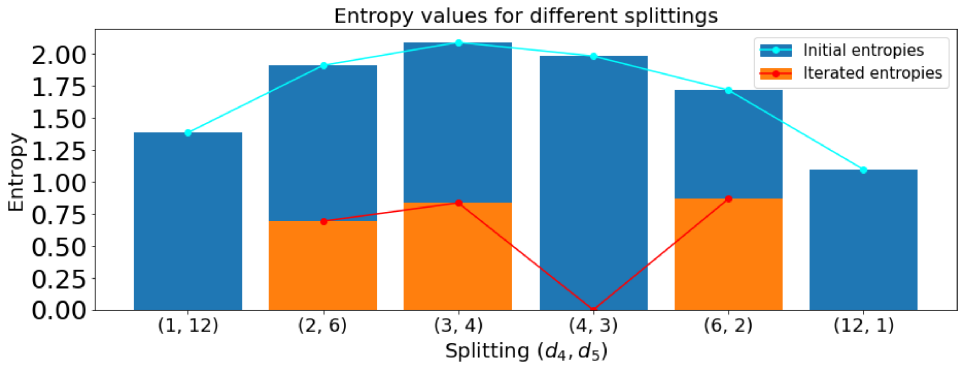

Figure 3.11: Bar plot with entropy evaluations and reduced optimization for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d'_1 = 4$ $d'_2 = 9$ $d'_3 = 6$	0.0	0.0	24.42	(800) 2000

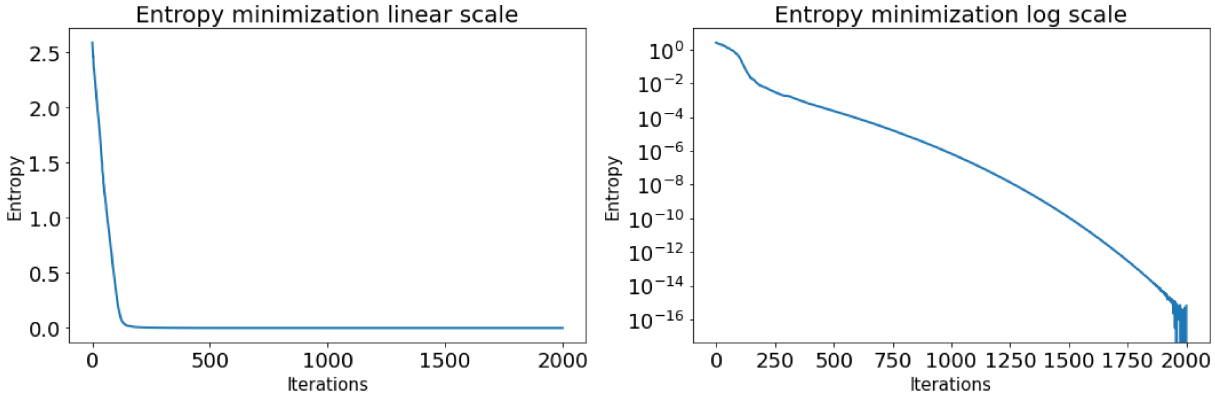

Figure 3.12: Entropy minimization as a function of the iterations

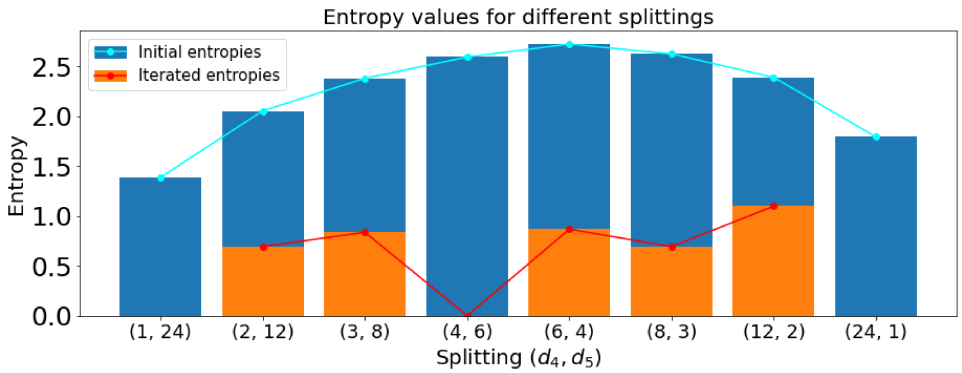

Figure 3.13: Bar plot with entropy evaluations and reduced optimization for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d'_1 = 8$ $d'_2 = 9$ $d'_3 = 9$	0.0	0.0	336.63	(1800) 10000

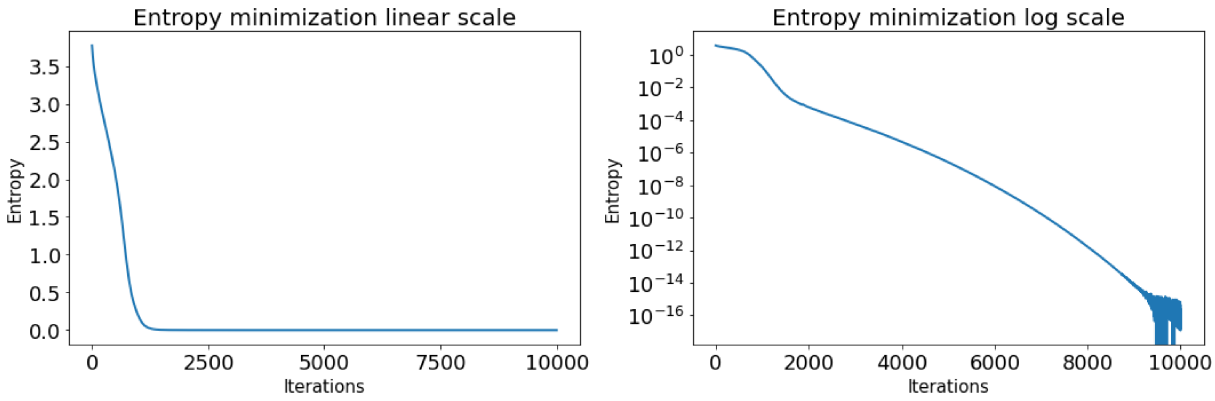

Figure 3.14: Entropy minimization as a function of the iterations

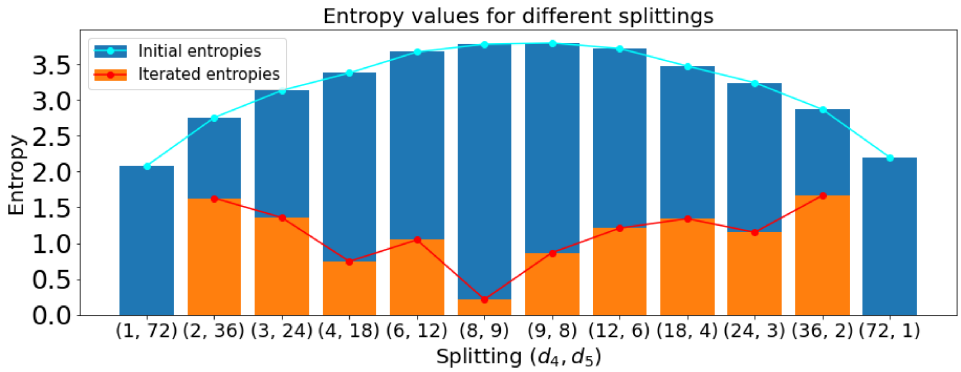

Figure 3.15: Bar plot with entropy evaluations and reduced optimization for every splitting.

- Case 3

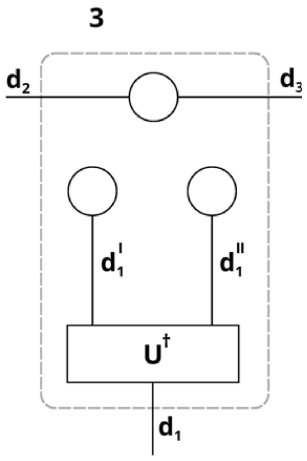

Figure 3.16: Benchmark case 3

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 3$	log(9)	0.0	2.26	1000

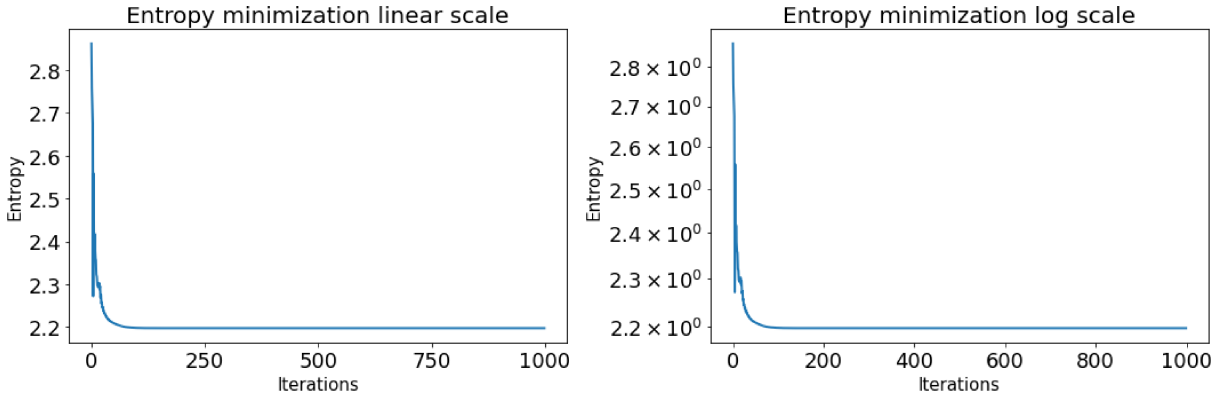

Figure 3.17: Entropy minimization as a function of the iterations

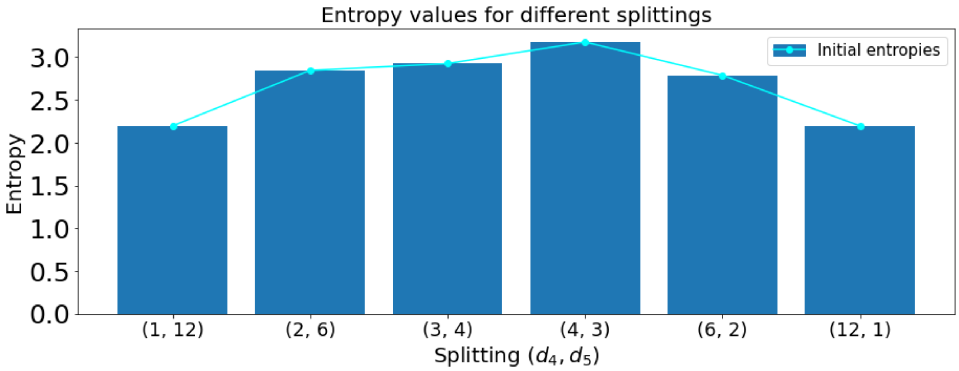

Figure 3.18: Bar plot with entropy evaluations for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d'_1 = 4\ d'_2 = 9\ d'_3 = 6$	log(9)	0.0	6.58	2000

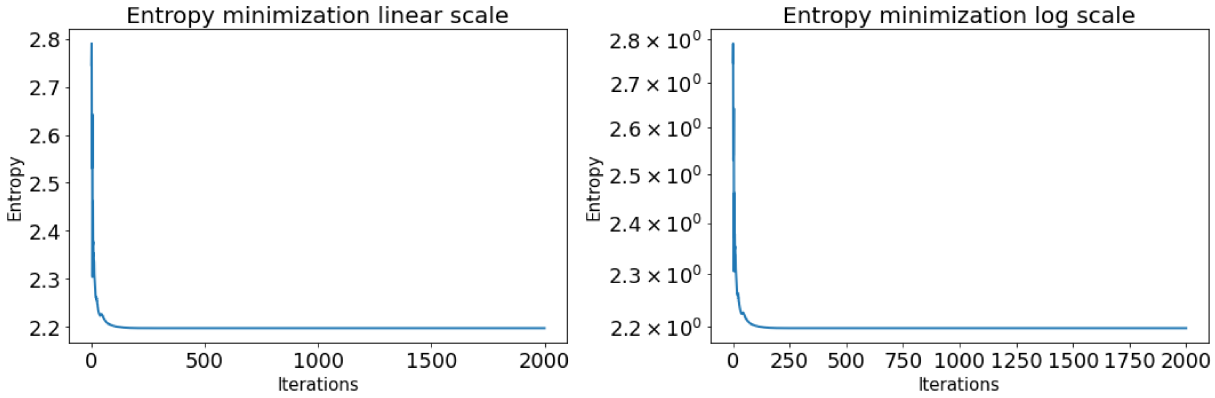

Figure 3.19: Entropy minimization as a function of the iterations

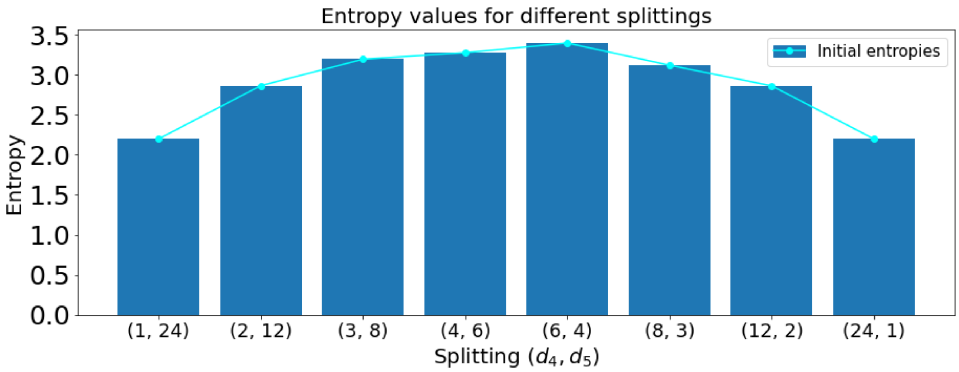

Figure 3.20: Bar plot with entropy evaluations for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 8\ d_2' = 9\ d_3' = 9$	log(9)	0.0	17.27	2000

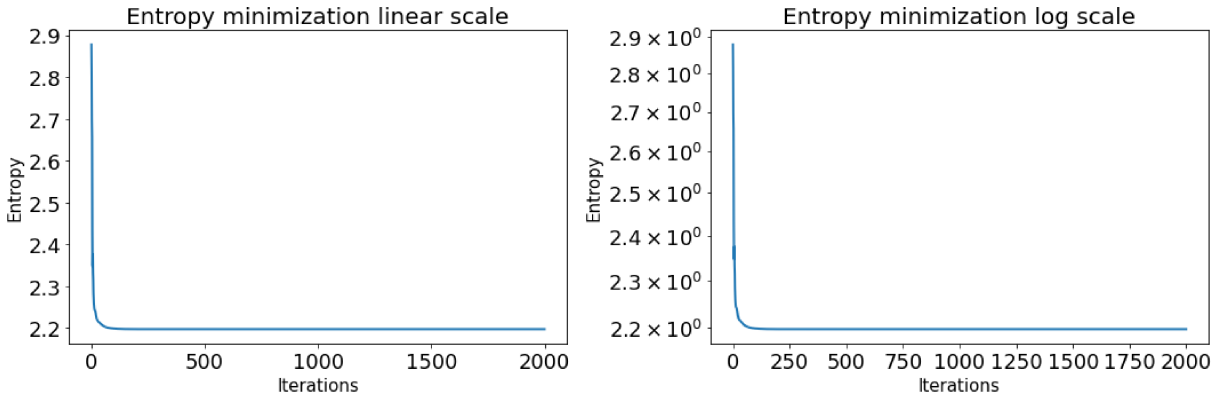

Figure 3.21: Entropy minimization as a function of the iterations

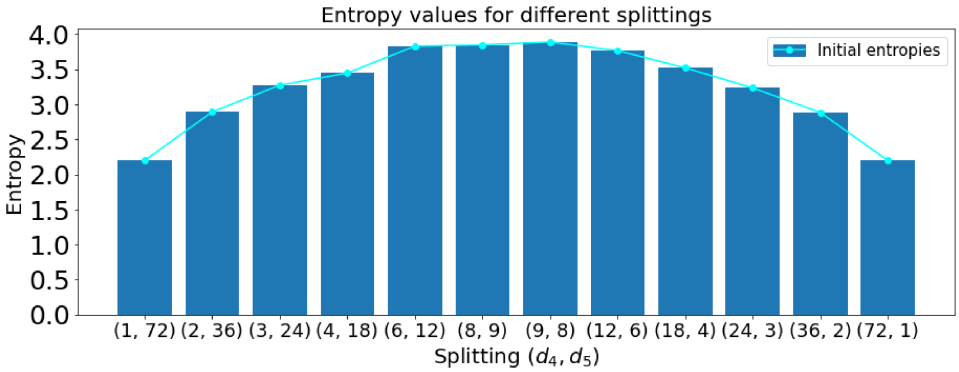

Figure 3.22: Bar plot with entropy evaluations for every splitting.

- Case 4

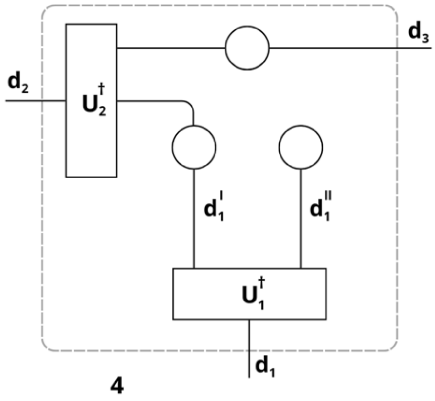

Figure 3.23: Benchmark case 4

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 3$	log(9)	0.0	6.87	(350) 1000

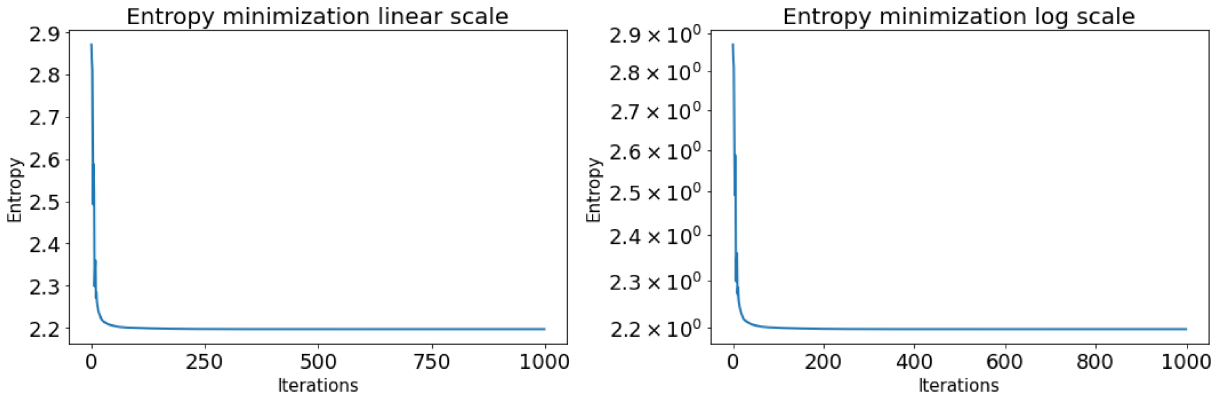

Figure 3.24: Entropy minimization as a function of the iterations

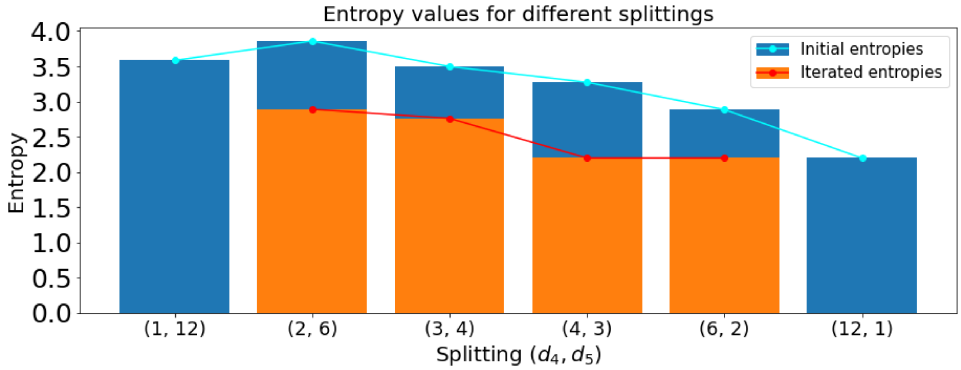

Figure 3.25: Bar plot with entropy evaluations and reduced optimization for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 6$	log(9)	0.0	36.30	(800) 2000

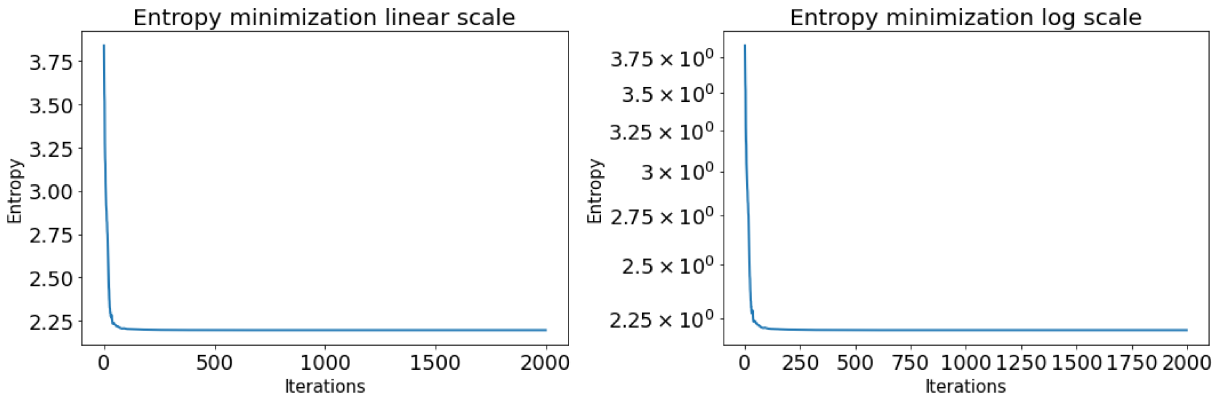

Figure 3.26: Entropy minimization as a function of the iterations

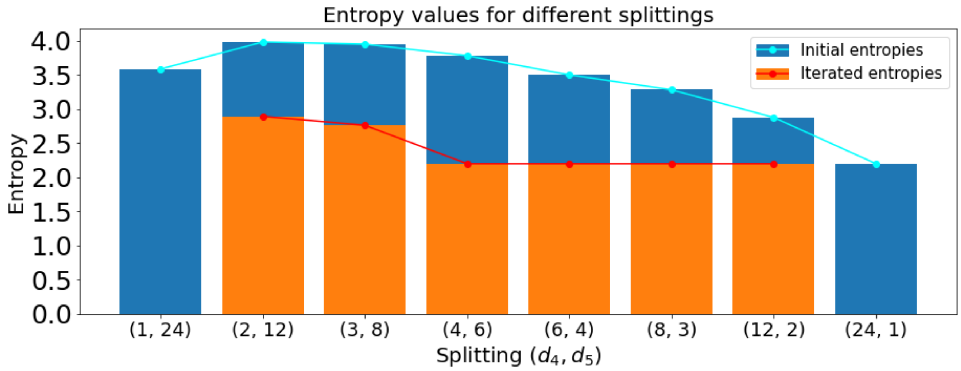

Figure 3.27: Bar plot with entropy evaluations and reduced optimization for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 8\ d_2' = 9\ d_3' = 9$	log(9)	0.0	575.05	(1800) 10000

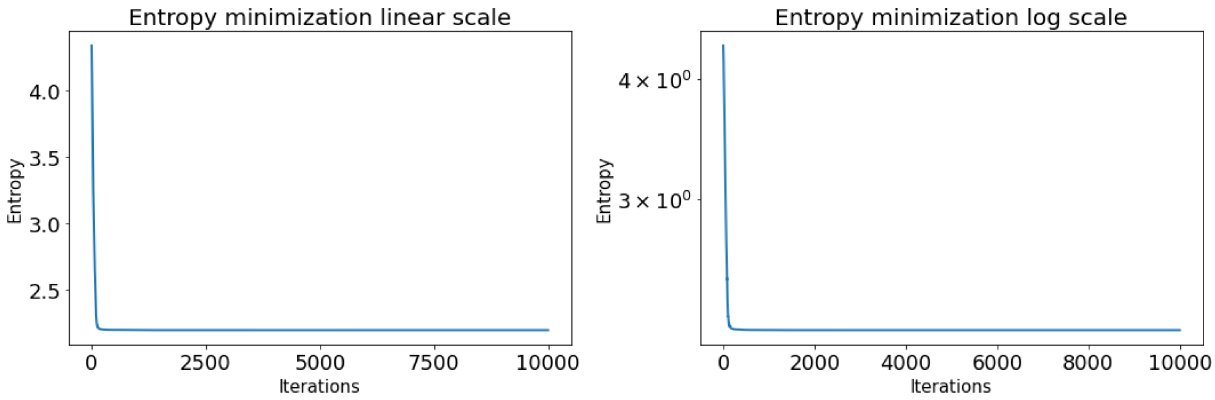

Figure 3.28: Entropy minimization as a function of the iterations

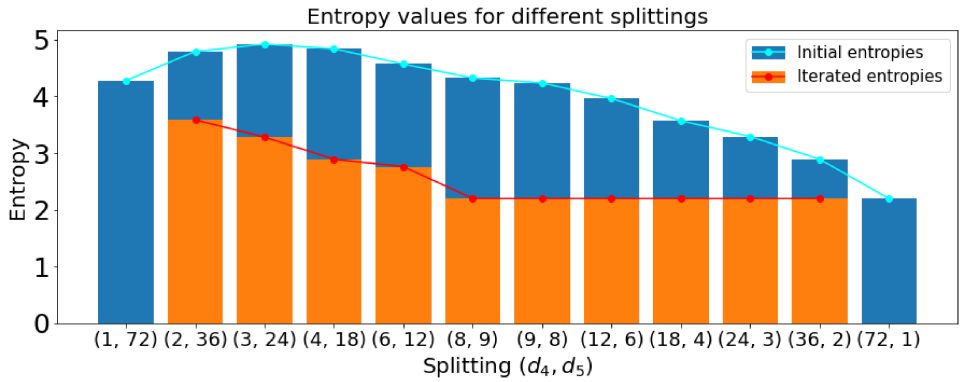

Figure 3.29: Bar plot with entropy evaluations and reduced optimization for every splitting.

- Case 5

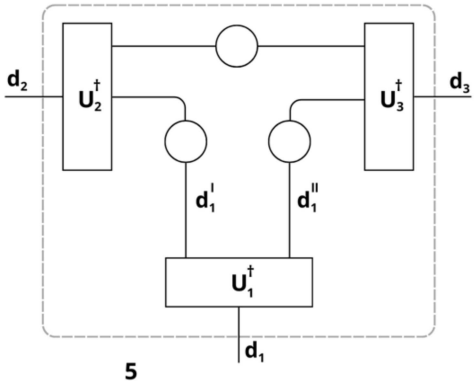

Figure 3.30: Benchmark case 5

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 3$	log(9)	0.0	13.01	(350) 1000

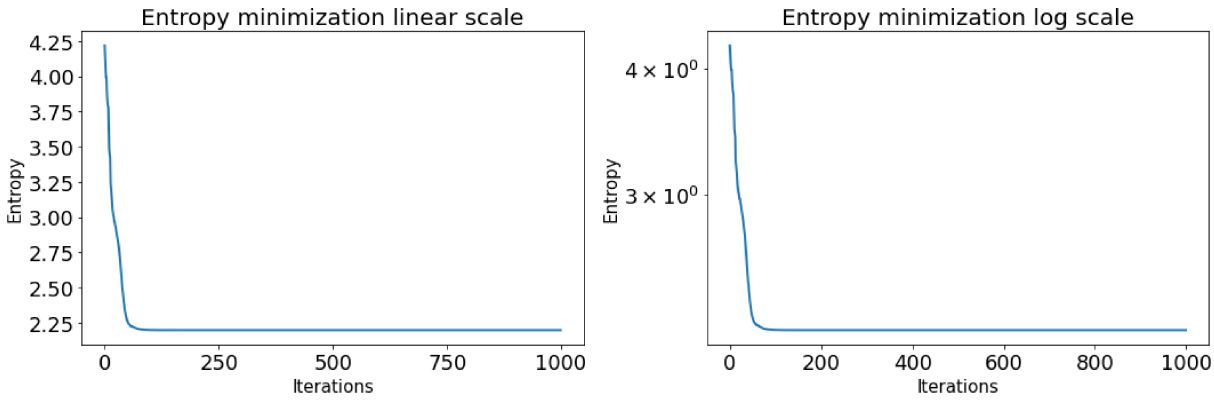

Figure 3.31: Entropy minimization as a function of the iterations

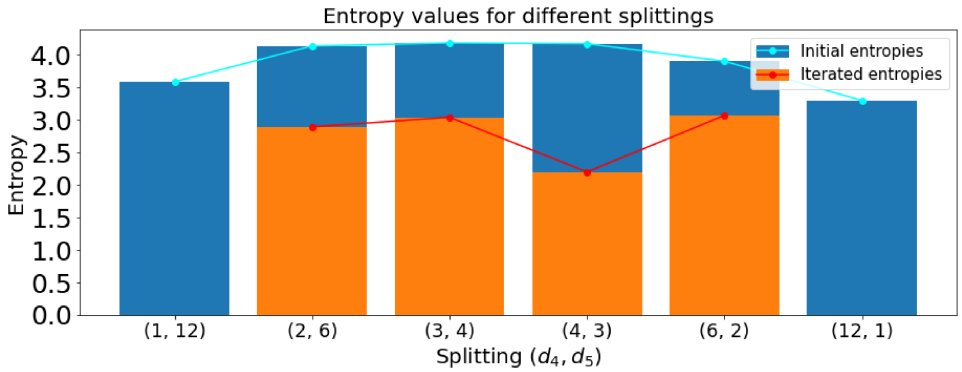

Figure 3.32: Bar plot with entropy evaluations and reduced optimization for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 4\ d_2' = 9\ d_3' = 6$	log(9)	0.0	109.15	(800) 2000

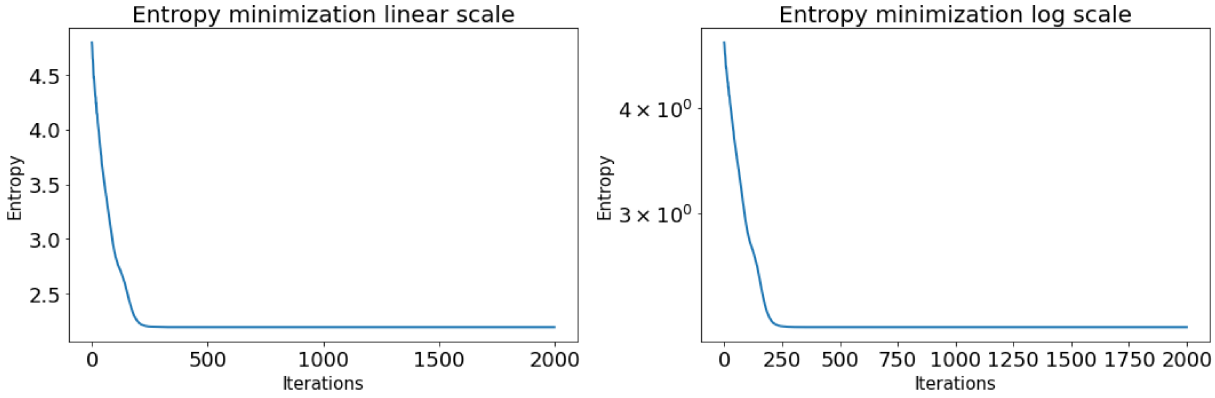

Figure 3.33: Entropy minimization as a function of the iterations

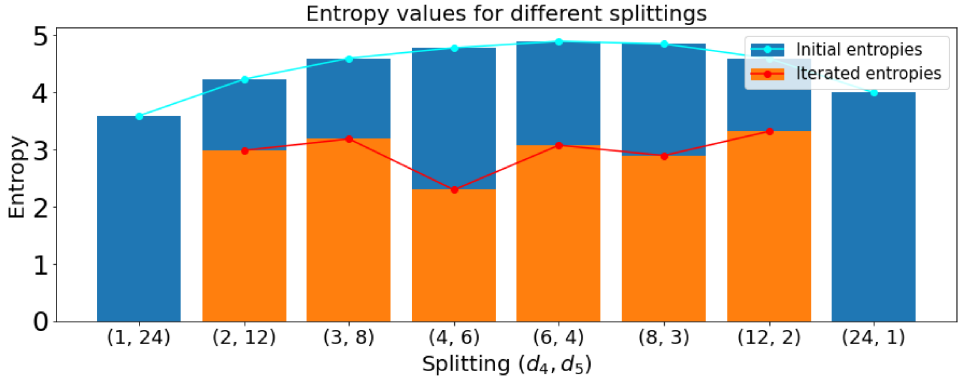

Figure 3.34: Bar plot with entropy evaluations and reduced optimization for every splitting.

Dimensions	Correct entropy	Discrepancy	CPU time [s]	(Reduced) Iterations
$d_1' = 8\ d_2' = 9\ d_3' = 9$	log(9)	$1.33 \cdot 10^{-15}$	6768.71	(2000) 10000

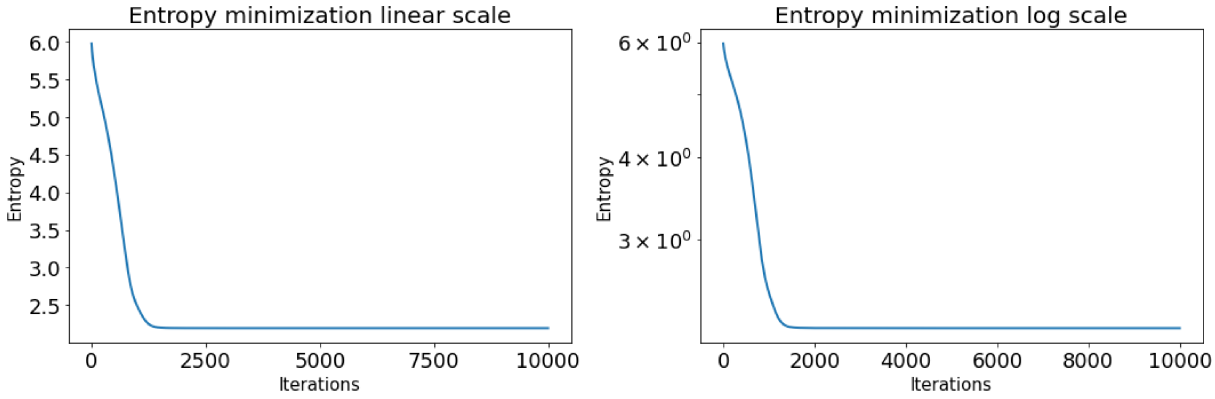

Figure 3.35: Entropy minimization as a function of the iterations

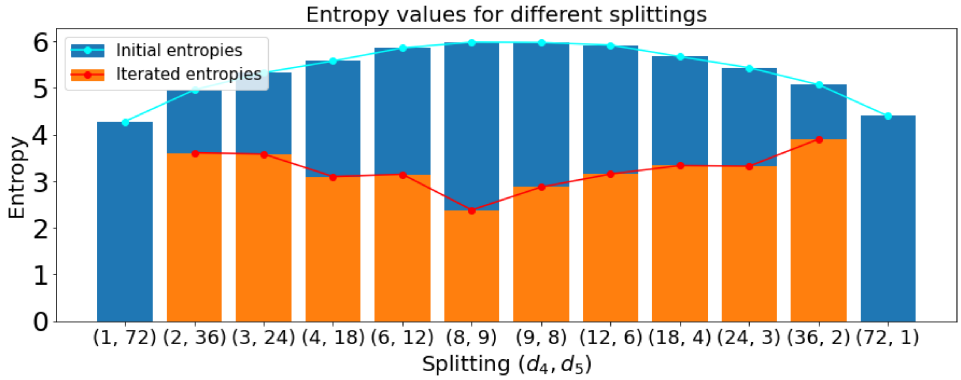

Figure 3.36: Bar plot with entropy evaluations and reduced optimization for every splitting.

3.5 Performance analysis and computational challenges

The optimization algorithm, developed as a direct search method using autodifferentiation with backpropagation, successfully meets the objectives. As shown by the results, carefully adjusting the algorithm's parameters is crucial for obtaining accurate evaluations. To improve accuracy while reducing computational time, the number of iterations and the learning rate must be chosen with care and expertise.

Cases 2, 4, and 5 require more computational time due to the complexity of identifying the correct underlying geometry, which involves multiple evaluations. Specifically, cases 4 and 5 are the most challenging because the dimensions of the final rank-3 tensor increase rapidly due to the multiplication of the underlying leg dimensions.

The most interesting case is Case 5, where larger dimensions are tested. This case represents the greatest achievement of the entanglement distribution algorithm, as the tensor under investigation displays the most complex underlying geometry with high leg dimensions. To ensure the algorithm convergence, a more sophisticated optimization approach is employed to optimize the unitary matrix. The large tensor size can cause the Singular Value Decomposition (SVD) to fail to converge because the resulting matrix, formed by contracting the tensor with the unitary matrix, becomes ill-conditioned. To mitigate this issue, different random seeds are used to initialize the unitary matrix in an attempt to avoid problematic entries. Despite these efforts, no tested seed configurations have completely resolved the problem. As a result, a small regularization term, $\epsilon = 1 \cdot 10^{-9}$, is added to the diagonal of the matrix to ensure SVD convergence. In contrast, the remaining cases (1, 3) are less complex and can be optimized more quickly due to their more manageable dimensions.

In all cases, measurable fluctuations in computational time and accuracy are observed across different runs. These fluctuations are attributed to background processes on the computer and the inherent

difficulty of achieving very small numerical values.

It is important to note that the required accuracy may be lower than the results presented here. When the algorithm is applied to larger loopy tensor networks, CPU time must be constrained and a threshold for accuracy should be established to balance the trade-off between approximation level and convergence.

Chapter 4

Conclusions

4.1 Comments on results

Loopy tensor networks can not be transformed into their unitary form using standard TN routines, due to the presence of cycles. This limitation prevents the use of efficient contraction methods necessary for eliminating false entanglement around loops. To address this, a protocol is developed to remove these spurious correlations and impose the unitary gauge in loopy tensor networks. Two approaches were analyzed: a direct search method and an AI-based neural network. For the former, several algorithms were tested, with autodifferentiation using backpropagation from the PyTorch library [40] ultimately being selected. For the latter, we are still investigating a machine learning setup which would enable the AI model to learn the features of the optimal distribution problem. This investigation is particularly challenging due to the difficulty of feeding the network with tensors of consistent shape. Consequently, the direct search method was chosen as the preferred approach for designing the algorithm to address this problem.

The entanglement distribution algorithm, implemented with autodifferentiation and backpropagation, was evaluated on five benchmark cases that represent the most probable tensor structures found in loopy tensor networks. The algorithm's performance was assessed based on the accuracy of the Von-Neumann entropy between left and right partitions of the tensor and the computational time required. Cases 1 and 3 were relatively straightforward to optimize, requiring minimal computational time. In contrast, Cases 2, 4, and 5 presented greater challenges due to the complexity of the underlying quantum system's geometry. Among these, Cases 4 and 5 were particularly demanding in terms of computational resources, as the rank-3 tensor dimensions quickly scale with the physical legs of the system. For Case 5, when the dimensions were high, careful tuning of the algorithm's parameters and random seed was required, along with a regularization term to ensure convergence of the SVD decomposition, as the matrix under decomposition became ill-conditioned. Overall, the algorithm proved to be fast and achieved a high level of accuracy with only a few iterations. In the context of loopy tensor networks, a trade-off between accuracy and computational time is necessary, with a lower accuracy threshold often acceptable to maintain computational efficiency. Therefore, the algorithm can be considered both efficient and flexible, allowing users to investigate various tensors by appropriately tuning the parameters.

The algorithm successfully identified the splitting of the leg d_1 into the legs d_4 and d_5 for all cases. The accuracy, measured as the discrepancy between the true Von-Neumann entropy of the tensor's left and right partitions and the evaluated entropy, approached machine precision. Consequently, the numerical discrepancy was effectively considered zero. The CPU time required for the algorithm scaled with the dimensions of the tensor; larger leg dimensions result in longer convergence times.

4.2 Outlook

To further optimize the algorithm and enhance its efficiency, many more pathways should be attempted, and careful research is necessary to fine-tune the learning rate and the number of iterations for the specific tensor under investigation. This ensures that the computational time and accuracy consistently meet the desired levels.

To fully harness the power of the algorithm, it is recommended to run the protocol on high-performance computing infrastructure, enabling the optimization of large tensors with dimensions comparable to those in quantum models.

The next step in this research is to develop a neural network capable of recognizing tensors and outputting the correct unitary matrix that decomposes the correlations. The completion of this research demands significant time and computational resources, as the dataset must cover a wide range of tensor shapes and underlying geometries. However, the expected benefit is substantial, potentially allowing for the rapid removal of spurious correlations across entire loopy tensor networks.

Bibliography

[1] Steven R. White. "Density matrix formulation for quantum renormalization groups". In: *Physical Review Letters* 69.19 (Nov. 1992), pp. 2863–2866. DOI: 10.1103/physrevlett.69.2863.

[2] Murg V. Verstraete F. and Cirac J. I. "Matrix product states, projected entangled pair states, and variational renormalization group methods for Quantum Spin Systems". In: *Advances in Physics* (2008). DOI: https://doi.org/10.1080/14789940801912366.

[3] J. Eisert, M. Cramer, and M. B. Plenio. "colloquium: Area laws for the entanglement entropy". In: *Reviews of Modern Physics* 82.1 (Feb. 2010), pp. 277–306. DOI: 10.1103/revmodphys.82.277.

[4] Vidal G. "Entanglement renormalization". In: *Physical Review Letters* (2007). DOI: https://doi.org/10.1103/physrevlett.99.220405.

[5] Román Orús. "A practical introduction to tensor networks: Matrix product states and projected entangled pair states". In: *Annals of Physics* 349 (Oct. 2014), pp. 117–158. DOI: 10.1016/j.aop.2014.06.013.

[6] J M Zhang and R X Dong. "Exact diagonalization: The bose–hubbard model as an example". In: *European Journal of Physics* 31.3 (Apr. 2010), pp. 591–602. DOI: 10.1088/0143-0807/31/3/016.

[7] Leo P. Kadanoff. "More is the same; phase transitions and mean field theories". In: *Journal of Statistical Physics* 137.5–6 (Sept. 2009), pp. 777–797. DOI: 10.1007/s10955-009-9814-1.

[8] Charlotte Froese Fischer. "General Hartree-Fock program". In: *Computer Physics Communications* 43.3 (Feb. 1987), pp. 355–365. DOI: 10.1016/0010-4655(87)90053-1.

[9] Nathan Argaman and Guy Makov. "Density functional theory: An introduction". In: *American Journal of Physics* 68.1 (Jan. 2000), pp. 69–79. DOI: 10.1119/1.19375.

[10] Vitalii L Ginzburg. "Certain theoretical aspects of radiation due to superluminal motion in a medium". In: *Soviet Physics Uspekhi* 2.6 (1960). DOI: 10.1070/pu1960v002n06abeh003185.

[11] Joel E. Moore. "The birth of topological insulators". In: *Nature* 464.7286 (Mar. 2010), pp. 194–198. DOI: 10.1038/nature08916.

[12] Masatoshi Sato and Yoichi Ando. "Topological superconductors: A Review". In: *Reports on Progress in Physics* 80.7 (May 2017), p. 076501. DOI: 10.1088/1361-6633/aa6ac7.

[13] Horst L. Stormer. "Nobel lecture: The fractional quantum hall effect". In: *Reviews of Modern Physics* 71.4 (July 1999), pp. 875–889. DOI: 10.1103/revmodphys.71.875.

[14] Lucile Savary and Leon Balents. "Quantum spin liquids: A Review". In: *Reports on Progress in Physics* 80.1 (Nov. 2016), p. 016502. DOI: 10.1088/0034-4885/80/1/016502.

[15] Malvin H. Kalos and Paula A. Whitlock. "Monte Carlo Methods". In: (Sept. 2008). DOI: 10.1002/9783527626212.

[16] W. M. Foulkes et al. "Quantum Monte Carlo simulations of Solids". In: *Reviews of Modern Physics* 73.1 (Jan. 2001), pp. 33–83. DOI: 10.1103/revmodphys.73.33.

[17] E. Y. Loh et al. "Sign problem in the numerical simulation of many-electron systems". In: *Physical Review B* 41.13 (May 1990), pp. 9301–9307. DOI: 10.1103/physrevb.41.9301.

[18] Kenneth G. Wilson. "The renormalization group: Critical phenomena and the Kondo problem". In: *Reviews of Modern Physics* 47.4 (Oct. 1975), pp. 773–840. DOI: 10.1103/revmodphys.47.773.

[19] Stellan Östlund and Stefan Rommer. "Thermodynamic limit of density matrix renormalization". In: *Physical Review Letters* 75.19 (Nov. 1995), pp. 3537–3540. DOI: 10.1103/physrevlett.75.3537.

[20] Ulrich Schollwöck. "The density-matrix renormalization group in the age of matrix product states". In: *Annals of Physics* 326.1 (Jan. 2011), pp. 96–192. DOI: 10.1016/j.aop.2010.09.012.

[21] Román Orús. "A practical introduction to tensor networks: Matrix product states and projected entangled pair states". In: *Annals of Physics* 349 (Oct. 2014), pp. 117–158. DOI: 10.1016/j.aop.2014.06.013.

[22] Subir Sachdev. "Tensor networks—a new tool for old problems". In: *Physics* 2 (Oct. 2009). DOI: 10.1103/physics.2.90.

[23] J. Eisert. "Entanglement and Tensor Network states". In: *arXiv.org* (Sept. 2013). URL: https://doi.org/10.48550/arXiv.1308.3318.

[24] Román Orús. "Advances on tensor network theory: Symmetries, fermions, entanglement, and holography". In: *The European Physical Journal B* 87.11 (Nov. 2014). DOI: 10.1140/epjb/e2014-50502-9.

[25] F. Verstraete, V. Murg, and J.I. Cirac. "Matrix product states, projected entangled pair states, and variational renormalization group methods for Quantum Spin Systems". In: *Advances in Physics* 57.2 (Mar. 2008), pp. 143–224. DOI: 10.1080/14789940801912366.

[26] Shi-Ju Ran et al. "Lecture notes of tensor network contractions". In: *arXiv.org* (July 2019). URL: https://arxiv.org/abs/1708.09213.

[27] Guifré Vidal. "Efficient classical simulation of slightly entangled quantum computations". In: *Physical Review Letters* 91.14 (Oct. 2003). DOI: 10.1103/physrevlett.91.147902.

[28] Andrew J. Ferris. "Area law and real-space renormalization". In: *Physical Review B* 87.12 (Mar. 2013). DOI: 10.1103/physrevb.87.125139.

[29] Mark Srednicki. "Entropy and area". In: *Physical Review Letters* 71.5 (Aug. 1993), pp. 666–669. DOI: 10.1103/physrevlett.71.666.

[30] G. Vidal et al. "Entanglement in quantum critical phenomena". In: *Physical Review Letters* 90.22 (June 2003). DOI: 10.1103/physrevlett.90.227902.

[31] M. B. Plenio et al. "Entropy, entanglement, and area: Analytical results for Harmonic Lattice Systems". In: *Physical Review Letters* 94.6 (Feb. 2005). DOI: 10.1103/physrevlett.94.060503.

[32] Bela Bauer and Chetan Nayak. "Area laws in a many-body localized state and its implications for topological order". In: *Journal of Statistical Mechanics: Theory and Experiment* 2013.09 (Sept. 2013). DOI: 10.1088/1742-5468/2013/09/p09005.

[33] Pietro Silvi et al. "The Tensor Networks Anthology: Simulation Techniques for many-body quantum lattice systems". In: *SciPost Physics Lecture Notes* (Mar. 2019). DOI: 10.21468/scipostphyslectnotes.8.

[34] LAPACK. *Linear Algebra PACKage*. URL: https://www.netlib.org/lapack/.

[35] BLAS. *Basic Linear Algebra Subprograms*. URL: https://www.netlib.org/blas/.

[36] Simone Montangero. *Introduction to tensor network methods: Numerical simulations of low-dimensional many-body quantum systems*. Springer, 2018.

[37] Nelder J. A. and Mead R. "A simplex method for function minimization". In: *The Computer Journal* (1695). DOI: https://doi.org/10.1093/comjnl/7.4.308.

[38] Dixit A. Alam T. Qamar S. and Benaida M. "Genetic algorithm: Reviews, implementations, and applications". In: *arXiv* (2020). DOI: https://doi.org/10.48550/arXiv.2007.12673.

[39] Haar A. "Der massbegriff in der Theorie der Kontinuierlichen Gruppen". In: *The Annals of Mathematics* (1933). DOI: https://doi.org/10.2307/1968346.

[40] Pytorch. *Pytorch*. URL: https://pytorch.org/.