Using the Belief Propagation algorithm for finding Tensor Networks approximations of many-body ground states

Roy Elkabetz

Using the Belief Propagation algorithm for finding Tensor Networks approximations of many-body ground states

Research Thesis

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Physics

Roy Elkabetz

Submitted to the Senate of
the Technion — Israel Institute of Technology
Adar 5780 Haifa March 2020

The research thesis was done under the supervision of Prof. Itai Arad in the Physics Department.

The generous financial support of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

Tensor Networks (TN) is a mathematical framework that enables an efficient description of many-body quantum states using a network of inter-connected tensors. In the last decade, there has been significant progress in using tensor networks in many fields such as condensed matter, cosmology, quantum information, machine learning, etc. One of the main challenges in the tensor networks community is the contraction of networks in high spatial dimensions (>1D), which can take exponential time. This presents critical issues in tensor networks optimizations, such as finding a tensor network approximation to a ground state of a many-body Hamiltonian. Over the years, many tensor networks algorithms have been developed to deal with this problem; all have their strengths and weaknesses. In this work, I will present a new approach for dealing with tensor networks contractions. It is based on a well-known method from statistical mechanics and computer science, which is called Belief Propagation (BP). Originally, this method was developed for approximating the marginals of many-body classical Gibbs distributions, or, more generally, of Probabilistic Graphical Models (PGM). As both problems involve a summation over an exponential number of configurations, it follows that the BP algorithm can be imported to the quantum setup. To that aim, I will introduce the Double-Edge Factor Graphs (DEFG), a new type of probabilistic graphical model that supports the quantum behavior of tensor networks. Based on this connection, I will present the Belief Propagation Update (BPU) algorithm for the contraction of tensor networks and compare it with a well-known tensor networks algorithm called the Simple Update (SU) algorithm on a 4x4 and 10x10 Antiferromagnetic Heisenberg model. As a final step, I will give a mathematical proof of the equivalence between BP and the trivial-SU, which is a variation of the SU algorithm and closely relates to the canonical representation in tensor networks.

Abbreviations and Notations

$\|\Psi\rangle$	:	A quantum state
ρ	:	A quantum state density matrix
TN	:	Tensor Networks
PGM	:	Probabilistic Graphical Models
FG	:	Factor Graphs
DEFG	:	Double-Edge Factor Graphs
RG	:	Region Graph
BP	:	Belief Propagation
BPU	:	Belief Propagation Update
GBP	:	Generalized Belief Propagation
SU	:	Simple Update
FU	:	Full Update
PSD	:	Positive Semi-Definite
SVD	:	Singular Value Decomposition

Chapter 1

Introduction

Tensor Networks and Probabilistic Graphical Models are two distinct fields that use graphical representation of systems with a large number of degrees of freedom in order to understand the local structure of systems and the relations between different local subsystems. In the last decade, tensor networks were mostly used in physics-based fields such as condensed matter physics [2][3][4][5][6] , statistical physics [7][8], quantum information [9][10], wherein the last few years we are witnessing an increased usage in tensor networks methods also in computer sciences disciplines such as artificial intelligence, machine learning, and deep neural networks [11][12][13]. On the other hand, probabilistic graphical models have been used in a wide variety of disciplines, including statistical physics [14], signal processing, artificial intelligence [15], and digital communications [16]. In recent years there has been some research on the connection and similarities of these two worlds [17] with the intentions of importing methods and algorithms between one another.

A central open question in the tensor networks community is how to contract high dimensional networks efficiently. Exact contraction of these networks requires exponential computational resources, which make them intractable. In the last decade, there has been much research on that problem, which led to many tensor networks algorithms such as the Full Update, Fast Full Update, Loop Update, Simple Update, etc. All of these methods lay on the continuum between accuracy and efficiency. In the world of classical probabilistic graphical models, we face a similar problem. In order to calculate some local marginal of a given joint probability distribution, one would have to sum over an exponential number of parameters. A method that tackles this problem in probabilistic graphical models is called Belief Propagation. Belief Propagation is an itera-

tive message-passing algorithm used to evaluate local marginals of a given join probability distribution efficiently [1]. In this thesis, I will describe our work with the framework of Double-Edge Factor Graphs. This probabilistic graphical model supports the quantum behavior of tensor networks, and first introduced in Ref. [18]. I will show that we have been able to harness the Belief Propagation algorithm's power for evaluating high dimensional tensor networks contractions using double-edge factor graphs. With that ability, we developed the Belief Propagation Update algorithm for finding tensor networks approximations for ground states of many-body local Hamiltonians. I will also show that our method gives the same numerical approximation as the well known Simple Update algorithm over a set of different physical models. These results led us to the realization that the BP equations and the trivial-SU algorithm, which is a variation of the Simple Update algorithm, are actually equivalence, and we went on to prove that relation mathematically. This equivalence is not a coincidence and may have been anticipated, as both algorithms are exact on graphs with a tree-like topology.

This dissertation's structure is as follows: Chapter 2 includes a short review of tensor networks and introduces the main problem of contracting tensor networks in high dimensions. Chapter 3 briefly discusses the framework of factor graphs while then introducing the framework of double-edge factor graphs in detail, which then used throughout the rest of this work. Chapter 4 exploits the double-edge factor graphs connection to tensor networks to develop the belief propagation update algorithm. In Chapter 5, we show some numerical results of the Antiferromagnetic Heisenberg model ground-state energy calculated using the belief propagation update algorithm and compare them to results obtained with the commonly used simple update algorithm. In Chapter 6, we give a numerical explanation to the results obtained using belief propagation update by presenting the trivial simple update algorithm and its relation to the belief propagation equations. In Chapter 7, we give a mathematical proof to the equivalence of the trivial simple update and the belief propagation algorithms that sums up our work. Finally, in Chapter 8, we talk about some open questions and possible improvements for the belief propagation update algorithm with the usage of the region-graphs method and the generalized belief propagation algorithm from the work of Yedida et al. in Ref. [1] and then conclude.

Chapter 2

Tensor Networks (TN)

2.1 What are Tensor Networks

Tensor Networks (TN) is a mathematical framework that enables an efficient description of many-body quantum states using a network of inter-connected tensors. The graphical language associated with TN makes complicated mathematical expressions intuitive and easy to manipulate. In the last decade, there has been significant progress in the usage of TN in a large number of fields such as condensed matter physics, statistical mechanics, quantum information, etc. [2][3][4][5][7][8][9][10]. There are several kinds of TN. In this work, we will focus mostly on MPS and PEPS, which are the 1D and 2D lattice-like TN. For a broader point of view on TN, one may refer to Refs. [19, 8].

In the framework of TN we deal with sets of tensors, where each tensor is a multi-dimensional array of complex numbers. To describe a tensor mathematically, we will use a single letter with indices, where each index corresponds to a specific dimension of the tensor. For example, T_{ijk} is a rank-3 tensor where the i, j, k indices correspond to its first, second and third dimensions. There exists an elegant graphical notation to describe a set of tensors and their inter connections. In this notation, a tensor is described by a diagram of a node with edges (legs), where each leg corresponds to a particular index. For example, a rank-1 tensor has a single index, hence it is a vector, its diagram shown in Fig. 2.1 is given by a node with a single leg. A matrix, which is a rank-2 tensor, is denoted by a diagram of a node with two legs (one for each dimension), as shown in Fig. 2.1. Now, let us define the notion of *tensor-contraction*, which takes two tensors and construct a third.

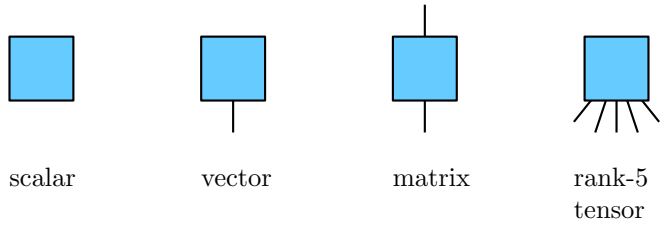


Figure 2.1: Examples of single tensor diagrams. From left to right: scalar, vector, matrix, rank-5 tensor.

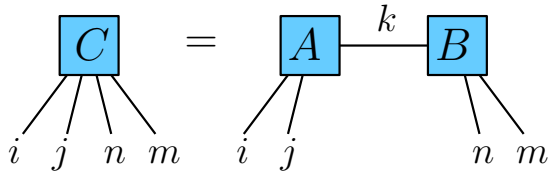


Figure 2.2: Example of the tensors contraction in Eq. (2.1).

Definition 2.1.1 (tensor-contraction) *Given two tensors $A_{i_1,\ldots,i_n}$ and $B_{j_1,\ldots,j_m}$, assume that the index i_k of A and the index j_ℓ of B run over the same range. Then the contraction of A, B along the i_k, j_ℓ indices is a new tensor defined by*

$$C_{\underbrace{i_1,\ldots,i_n}_{\text{no } i_k},\underbrace{j_1,\ldots,j_m}_{\text{no } j_\ell}} \stackrel{\text{def}}{=} \sum_t A_{i_1,\ldots,t,\ldots,i_n} \cdot B_{j_1,\ldots,t,\ldots j_m}$$

where on the right-hand-side above, the index t is in place of i_k and j_ℓ.

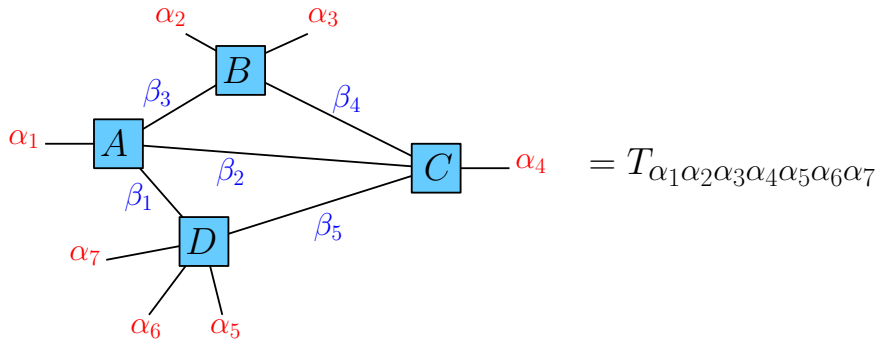
For example: given the two rank-3 tensors A_{ijk} and B_{knm}, the contraction along their common index k can be mathematically expressed as

$$C_{ijnm} = \sum_k A_{ijk} B_{knm} \tag{2.1}$$

while its tensor network diagram is shown in Fig. 2.2. Next, using tensors contractions we can define the notion of *tensor-network*.

Definition 2.1.2 (Tensor Network)
A Tensor Network is a set of tensors together with a contraction relation between their indices so that their overall contraction produces a single global tensor. Graphically, it is denoted as a graph G in which every node corresponds to a tensor in the network and its adjacent edges correspond its indices. Edges that connect two nodes imply a contraction of the two tensors along these indices, whereas 'open edges', which are adjacent to a single node, denote indices that are not contracted, and therefore appear as part of the global tensor that is represented by the network.

For example: the following tensor network



corresponds to the tensor contraction

$$T_{\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7} = \sum_{\beta_1\beta_2\beta_3\beta_4\beta_5} A_{\alpha_1\beta_3\beta_2\beta_1} B_{\beta_3\alpha_2\alpha_3\beta_4} C_{\beta_4\alpha_4\beta_5\beta_2} D_{\beta_5\alpha_5\alpha_6\alpha_7\beta_1} \tag{2.2}$$

Having some basic understanding of TN lets see how they relate to quantum states representations.

2.1.1 Quantum states representations using Tensor Networks

Given a quantum system of N spins with local Hilbert space dimension d, the system's quantum state $|\Psi\rangle$ can be generally expressed as

$$|\Psi\rangle = \sum_{i_1,i_2,\ldots i_N} C_{i_1,i_2,\ldots,i_N} |i_1\rangle|i_2\rangle\ldots|i_N\rangle \tag{2.3}$$

where $C_{i_1,i_2,\ldots i_N}$ is a rank-N tensor that contains d^N complex parameters which are the weights of all d^N different spins configurations in $|\Psi\rangle$. The summation is taken over all $\{i_1, i_2, \ldots, i_N\}$ configurations. The main problem of exactly simulating such a quantum state on a classical computer is its exponential space complexity in the system size. For example, a spin-1/2 system of 32 spins will have 2^{32} configurations which is the maximum amount of memory that can be addressed by a pointer on a 32-bit classical computer. Therefore, even a moderate system of 100 spins would be impossible to exactly simulate on any kind of classical computer in the near future. Here, the framework of TN comes to our help. As we will see, there are certain classes of quantum states in which the tensor $C_{i_1,i_2,\ldots,i_N}$ can be well approximated by a network of smaller interconnecting tensors with a total number of network parameters which is polynomial in the system size. In Fig. 2.3 one can see four different examples of such tensor

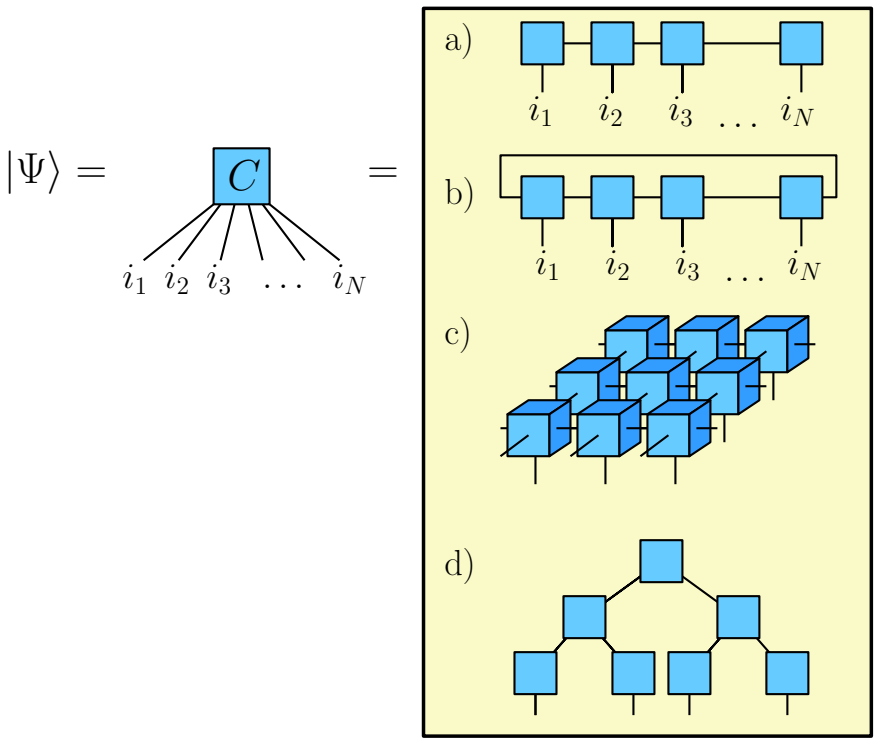


Figure 2.3: Examples of different types of TN representations for the $C_{i_1,i_2,\ldots i_N}$: (a) Matrix Product States (MPS) with open b.c. (b) MPS with periodic b.c. (c) Projected Entangled Pair States (PEPS). (d) Tree TN.

networks representations for $C_{i_1,i_2,\ldots i_N}$, where the first three are written as a contraction of $\mathcal{O}(N)$ small tensors and the fourth as a contraction of $\mathcal{O}(N \log N)$. Assuming that each tensor has $\mathcal{O}(1)$ legs, which is independent on the systems size, the overall network only uses $\mathcal{O}(N)$ parameters to describe $C_{i_1,i_2,\ldots i_N}$. Thus, even systems of size $N = 1000$ could be easily represented on a classical computer. Let us look more closely at the first example in Fig. 2.3, the Matrix Product States (MPS) with open boundary conditions.

An MPS is a many-body quantum state $|\Psi\rangle$ that has a TN representation using a $1D$ array of tensors connected in a series, where each tensor associated with a single physical degree of freedom / spin. There are two kinds of MPS representations, finite and infinite, where the finite MPS could have open or periodic boundary conditions, shown in Fig. 2.4. Each MPS tensor (except for the boundary tensors of an open MPS) has three legs as depicted in Fig. 2.4-4: A single leg, which corresponds to the physical degree of freedom where in Fig. 2.4-4 associated with the index i, and two more legs associated with indices α, β, which correspond to some auxiliary degrees of freedom. Later we will see that those

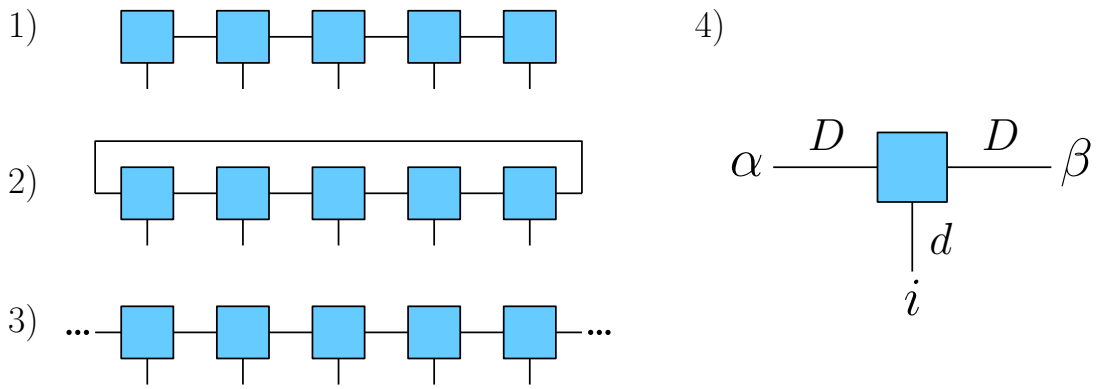


Figure 2.4: 5 sites MPS examples: (1) Open boundary conditions. (2) Periodic boundary conditions. (3) Infinite MPS. 4) A single MPS tensor.

auxiliary degrees of freedom are highly related to the amount of entanglement between the physical degrees of freedom in the quantum state. Assuming that the range of these indices is $i \in 1, 2, \ldots, d$ and $\alpha, \beta \in 1, 2, \ldots, D$, notice that excluding boundaries, all the MPS tensors $T^i_{\alpha\beta}$ contain dD^2 complex parameters. When d, D are independent of N this implies that the N spins quantum state is represented using only $\mathcal{O}(NdD^2)$ parameters, which is linear in the system's size. We remark that the value of the parameters d, D is often referred as *bond dimension*. More specifically, d referred as the *physical bond dimension* while D referred as the *virtual bond dimension*. I would like to emphasize that given a quantum state description in the form of a rank-N tensor $C_{i_1,i_2,\ldots i_N}$, it could potentially have many different TN representations such as MPS, PEPS, MERA, etc. Each TN representation has its own pros and cons where for example MPS provides very accurate results in describing 1D gapped systems with states that have an area-law in an efficient way, while it totally fails to describe systems in higher dimensions. On the other hand for 2D gapped systems one can use the PEPS representation which provides accurate description of quantum states which satisfy the area-law. Though, to evaluate expectation values one would need to work through an exponential summation of parameters which even for small systems of 100 spins is already intractable. I will elaborate on that in the next few sections.

2.2 Schmidt Decomposition and Entanglement Entropy in Tensor Networks

There is a strong relation between the geometrical structure of the chosen TN and the entanglement between the different spins in the quantum many-body state it represents. Given some quantum state $|\Psi\rangle$, the Von-Neumann entropy is the quantum equivalent of classical entropy and is defined as

$$S(\rho) = -\mathrm{Tr}[\rho \log \rho]$$

where $\rho = |\Psi\rangle\langle\Psi|$ is the quantum state density matrix. Using the Von-Neumann entropy definition one can define the *entanglement entropy*. The *entanglement entropy* is a measure of the amount of quantum correlations between two parts of a quantum systems. Assume some bi-partitioning of $|\Psi\rangle$ into two disjoint parts A and A^C. The entanglement entropy between the two parts is equal to

$$S(\rho_A) = -\mathrm{Tr}[\rho_A \log \rho_A] = -\mathrm{Tr}[\rho_{A^C} \log \rho_{A^C}] = S(\rho_{A^C})$$

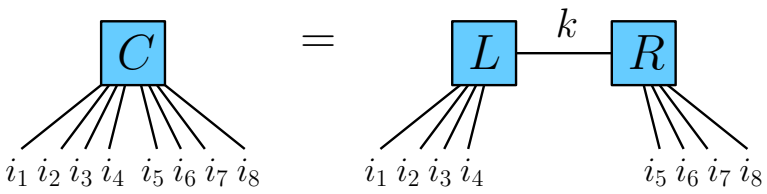
where $\rho_A = \mathrm{Tr}_{A^C}(\rho)$ is the reduced density matrix of subsystem A and $\rho_{A^C} = \mathrm{Tr}_A(\rho)$ is the reduced density matrix of subsystem A^C. The strictly positive eigenvalues λ_k^2 of both ρ_A and ρ_{A^C} correspond to the Schmidt values λ_k of the Schmidt decomposition of $|\Psi\rangle$ into A and A^C through

$$|\Psi\rangle = \sum_{k=1}^{D_{SR}} \lambda_k |\varphi_k^A\rangle |\phi_k^{A^C}\rangle$$

where D_{SR} is called the *Schmidt-Rank*. If the *Schmidt-Rank* of some quantum state partitioning is $D_{SR} = 1$ we say that the state is a *separable state*, else it is called an *entangled state* [20]. Next, let us see how this communicates with the framework of TN. We start by showing with an example that the TN representation virtual bond dimension determines the Schmidt rank between subsystems in the whole systems while also gives an upper bound on the entanglement entropy between these subsystems. Given some quantum state of $N = 8$ spins

$$|\Psi\rangle = \sum_{i_1,i_2,\ldots i_8} C_{i_1,i_2,\ldots,i_8} |i_1\rangle |i_2\rangle \ldots |i_8\rangle \tag{2.4}$$

Assume that $C_{i_1,i_2,\ldots,i_8}$ is represented using the TN



where the bond dimension of k is D. Explicitly, this means

$$C_{i_1,i_2,\ldots,i_8} = \sum_{k=1}^{D} L_{i_1 i_2 i_3 i_4 k} R_{k i_5 i_6 i_7 i_8} \tag{2.5}$$

Then,

$$|\Psi\rangle = \sum_{k=1}^{D} \left(\sum_{i_1,i_2,i_3,i_4} L_{i_1 i_2 i_3 i_4 k} |i_1\rangle|i_2\rangle|i_3\rangle|i_4\rangle \right) \otimes \left(\sum_{i_5,i_6,i_7,i_8} R_{k i_5 i_6 i_7 i_8} |i_5\rangle|i_6\rangle|i_7\rangle|i_8\rangle \right) \tag{2.6}$$

$$= \sum_{k=1}^{D} |L^k\rangle \otimes |R^k\rangle \tag{2.7}$$

Looking at the equality in Eq. (2.7), we can express $|\Psi\rangle$ as the superposition of D product states. The product states are defined by the partitioning of the quantum state into two subsystems $\{i_1, i_2, i_3, i_4\}$ and $\{i_5, i_6, i_7, i_8\}$ along the leg associated with the index k. Therefore, the *Schmidt rank* of this partitioning is at most D which is the bond dimension of k. The reduced density matrix of the $\{i_1, i_2, i_3, i_4\}$ subsystem is given by

$$\rho_L = \mathrm{Tr}_R \left(|\Psi\rangle\langle\Psi| \right) = \sum_{k,k'} X_{kk'} |L^k\rangle\langle L^{k'}| \tag{2.8}$$

where $X_{kk'} = \langle R^k | R^{k'} \rangle$. Then, the entanglement entropy of $|\Psi\rangle$ for this partitioning is $S(\rho_L) = -\mathrm{Tr}(\rho_L \log \rho_L)$ which is clearly upper bounded by $\log(D)$ because $X_{kk'}$ has at most D different eigenvalues that are not zero. Thus, we see that the geometrical structure of the chosen TN determines the Schmidt rank and entanglement entropy between parts of the system.

Now, let us see how the Schmidt rank affects the possible TN structure. Assume we have a quantum state given in the next Schmidt decomposition:

$$|\Psi\rangle = \sum_{i_1,i_2,\ldots i_8} C_{i_1,i_2,\ldots,i_8} |i_1\rangle|i_2\rangle\ldots|i_8\rangle = \sum_{k=1}^{D} \lambda_k |L^k\rangle \otimes |R^k\rangle \tag{2.9}$$

This implies we can write the tensor $C_{i_1,i_2,\ldots,i_8}$ as the contraction over the next

three tensors $L^{k}_{i_1,i_2,i_3,i_4}, R^{k'}_{i_5,i_6,i_7,i_8}$ and $\lambda_{kk'} = \lambda_k \delta_{kk'}$ as follows:

$$C_{i_1,i_2,\ldots,i_8} = \sum_{k,k'} \lambda_{kk'} L^{k}_{i_1,i_2,i_3,i_4} R^{k'}_{i_5,i_6,i_7,i_8} \tag{2.10}$$

Eq. (2.10) can be graphically illustrated using the TN diagram shown in Fig. 2.5. Thus, k, k' could be referred via a single index with bond dimension D.

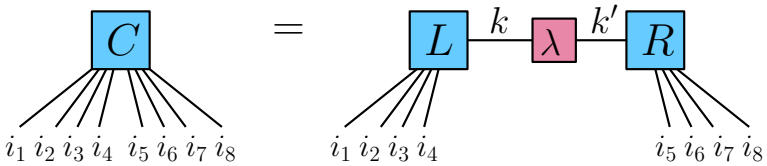


Figure 2.5: A TN representation of Eq. (2.10).

We see that a low Schmidt rank implies a tensor network with a low bond dimension D, corresponding to that particular bi-partition.

From the above, we can conclude that the geometrical structure of a TN directly upper-bounds the entanglement entropy and Schmidt rank of the partitioning and vice versa. Also, in the example above we used a TN with a single virtual leg k. The generalization for TN partitioned over a boundary of length $|\partial L|$ (many legs) with virtual bond dimensions D is straightforward. In that case, the entanglement entropy is upper bounded by $S(\rho_{\partial L}) = |\partial L| \log(D)$, see Fig. 2.6.

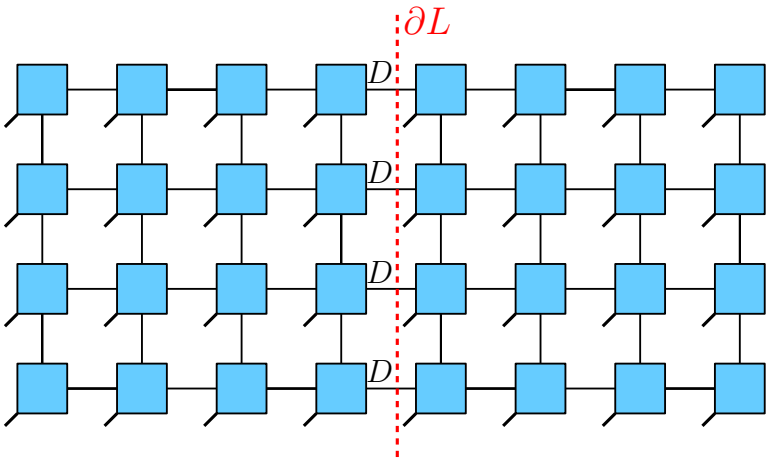


Figure 2.6: A TN partitioning through the boundary $|\partial L|$ forces an upper on the entanglement entropy of $S(\rho_{\partial L}) = |\partial L| \log(D)$.

2.3 The Entanglement Entropy Area Law in Tensor Networks

In the previous section, we have seen a close relationship between the TN geometrical structure and the Schmidt rank upper bound. Taking advantage of that relationship, it turns out that it is possible to choose quantum states TN representations that satisfy the *entanglement entropy area law* by construction. Those TN representations are beneficial in approximating quantum states observed in nature, which are often known to satisfy the entanglement entropy area law.

Given a quantum state TN representation $|\Psi\rangle$, we say that it satisfies the *entanglement entropy area law* if for every partitioning of it into two connected parts separated by a boundary ∂L, the entanglement entropy is upper bounded by $S_{\partial L} \leq c|\partial L|$, where $c = O(1)$ is some constant, independent of L. Less formally, it means that the entanglement entropy of a subset of spins scales as the contact area of the subset with the rest of the system, rather then its volume, as in typical quantum states. A classical example of it in 2D is illustrated in Fig. 2.7 for the case of PEPS with constant bond dimension D. The Schmidt rank in that case would be bounded by D^{4L}. Thus, the entanglement entropy would be upper-bounded by

$$S_{4L}(\Psi) \leq 4L \log(D) \tag{2.11}$$

which satisfies the area law.

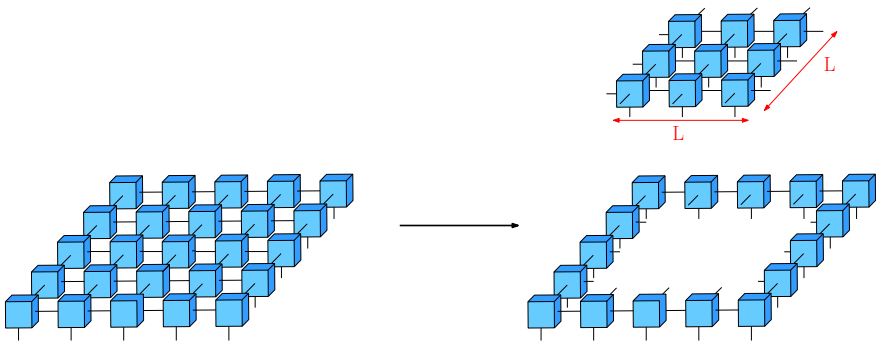


Figure 2.7: Splitting a 5×5 PEPS with bond dimension D into inner and outer subsets with boundary size $|\partial L| = 4L$.

The two most common TN representations that satisfy the area law are MPS and PEPS which are used for describing 1D and 2D gapped spin systems respectively. Next, let us see how we can use TN to extract information about a system such as calculating expectation values via TN contraction.

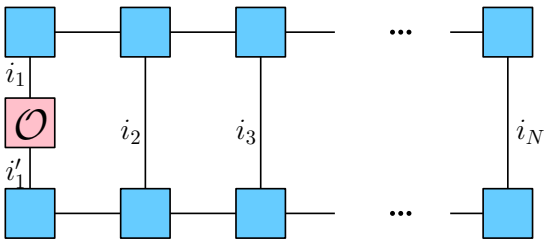


Figure 2.8: MPS expectation value of the local observable O on spin i_1.

2.4 Contracting Tensor Networks

Given a TN, in order to calculate some expectation value one would have to contract all the indices in the network. The order at which one follows to contract the indices does not have any effect on the final result, although it could highly affect the calculation total computational cost. A contraction order computational cost is evaluated by counting the number of operations that must be taken to sum over all the contracted indices in a network. There is more than a single contraction order for a given network, where different orderings would mostly have different computational costs. Let us see how to evaluate the computational cost of a specific contraction. Consider the next two tensors, $A_{i_1,\ldots,i_k,j_1,\ldots,j_n}$ and $B_{i_1,\ldots,i_k,\ell_1,\ldots,\ell_m}$, where each index in both tensors can take D different values. Then, tensor $A_{i_1,\ldots,i_k,j_1,\ldots,j_n}$ is of the size D^{k+n} and $B_{i_1,\ldots,i_k,\ell_1,\ldots,\ell_m}$ is of size D^{k+m}. Notice that both tensors have k common indices, $i_1,\ldots,i_k$. The contraction computational cost of A with B is calculated by iterating over every index once, both for common and individual indices. One who is familiar with basic linear algebra can think about this process as some sort of generalized dot product where instead of vectors which are rank-1 tensors we are dealing with tensors with any rank. Thus, the contraction can be written as

$$C_{j_1,\ldots,j_n,\ell_1,\ldots,\ell_m} = \sum_{i_1,\ldots,i_k} A_{i_1,\ldots,i_k,j_1,\ldots,j_n} B_{i_1,\ldots,i_k,\ell_1,\ldots,\ell_m} \tag{2.12}$$

The computation of the above sum requires $\mathcal{O}\left(D^{k+n+m}\right)$ algebraic operations of multiplication and summation, and therefore we say that the computational cost of these two tensors contraction is $\mathcal{O}\left(D^{k+n+m}\right)$. Next, assume we have a quantum state $|\Psi\rangle$ represented by an MPS and we would like to calculate the local observable O expectation value on the first spin i_1. The relevant network is illustrated in Fig. 2.8.

The simplest algorithm to do that is to choose a contraction order of tensors

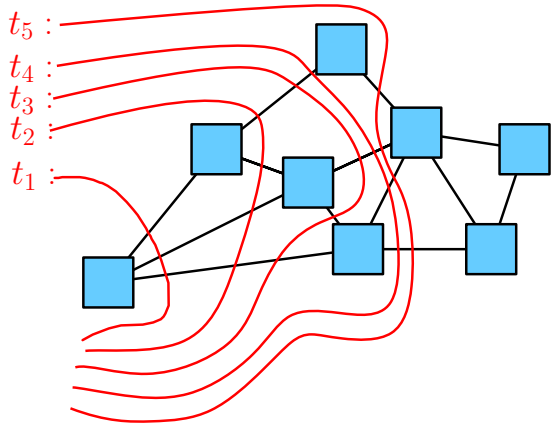


Figure 2.9: Random TN contraction using the bubble method. We start with the most left tensor at time step t_1 and advance through the network step by step from left to right swallowing a single tensor at a time.

and then contracting them one by one following that order. At each step we store a tensor which is equal to the contraction of all the tensors from previous steps and contract it with the next tensor in line. One can imagine this process as a "bubble" which swallows the network step by step until there are no tensors left. After contracting all the indices in the network we are left with a scalar which is the wanted expectation value. An illustration of the "bubble" process on a random TN is shown in Fig. 2.9

As we mentioned before, different contraction orderings would have different computational costs, i.e. we can choose two main different orderings to contract an MPS: vertically or horizontally, both are illustrated in Fig. 2.10. The horizontal contraction would have a total computational cost of $\mathcal{O}\left(NdD^4\right)$ which is linear in the system's size while the vertical one will have $\mathcal{O}\left((dD)^N\right)$ which is exponential in the system's size. We see that in the case of contracting an MPS there is a great deal in choosing the contraction order in the right way, where one could avoid an exponential computational cost by contracting the tensor network horizontally rather then vertically.

2.4.1 The problem in contracting 2D Tensor Networks

In the 1D case we have seen that there is a great deal in choosing the contraction order of the tensor network to avoid computational cost which is exponential in the system's size. It turns out that this is no longer the case when contracting networks in higher dimensions. For example, the most efficient contraction of a PEPS network is carried out by starting from a corner tensor and step by

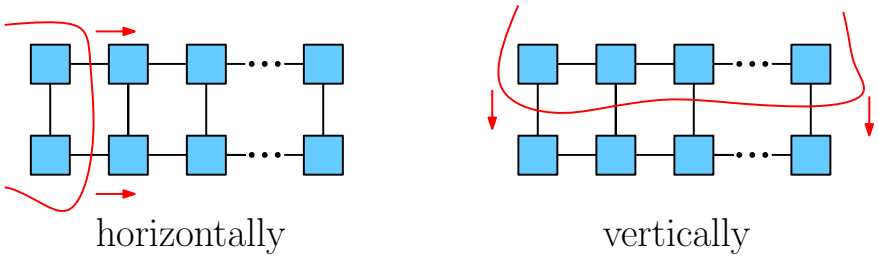


Figure 2.10: left: horizontal contraction of an MPS with a single step computational cost of $\mathcal{O}(dD^4)$ where d and D are the physical and virtual bond dimensions respectively. **right:** vertical contraction of an MPS with a single step computational cost of $\mathcal{O}\left((dD)^N\right)$.

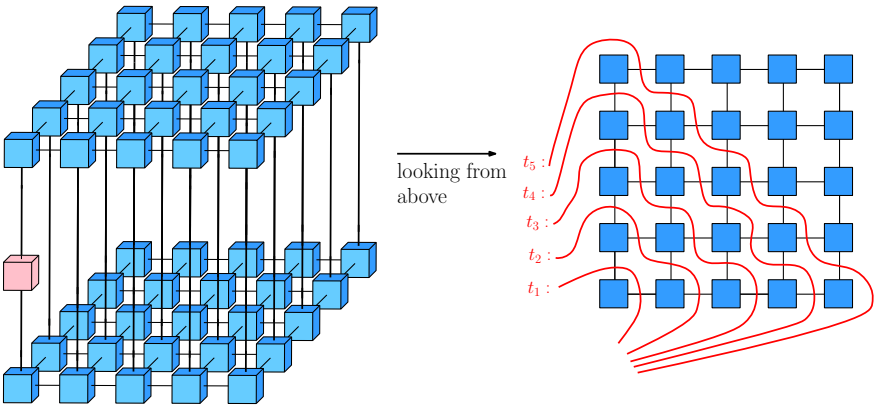


Figure 2.11: PEPS contraction example.

step advancing through the middle of the network into the other side. In the general case of N spins PEPS, no matter what is the chosen contraction order, eventually we will end up crossing $\mathcal{O}(N)$ legs of bond dimension D and so the computational cost of that step will be exponential in the system's size.

The exponential complexity of contracting TN in 2D and also higher dimensions causes an intractable calculation already for systems with $N = 100$. It can actually be shown that contracting a general 2D TN is classified as a #P-hard [21], which is harder then NP. A problem is said to be in the class NP if given an answer it could be verified in polynomial time. While the #P-hard class is the set of counting-problems associated with the problems in the NP class. Thus, a member in #P-hard would relate to the count of answers which could be verified in polynomial time. In order to deal with these kind of problems, there has been a lot of research in finding algorithms for approximating TN contractions in high dimensions such as CTM, TNR, bMPO, Single-Layer, etc [22, 19, 8]. Given a highly entangled quantum system, it is not obvious which of the methods above, if any, is the best one for calculating local observable expectations [21]. In the

next chapters, we would like to introduce a new method for dealing with this kind of problems using the framework of Probabilistic Graphical Models (PGM) and iterative message-passing algorithms. Specifically, we will use the Belief Propagation (BP) algorithm for calculating local marginals of a given joint probability distribution factorization and show how we can exploit it in contracting 2D TN like PEPS.

Chapter 3

Probabilistic Graphical Models (PGM)

Probabilistic Graphical Models (PGM) are a powerful framework that uses a graph-based representation to encode the set of independences of a multivariate probability distribution. In the last decade, there was tremendous growth in the usage of PGM in a wide range of applications such as computer vision, natural language processing, machine learning, etc. [11][12][13][15]. In this section I will give a quick overview of Factor Graphs (FG), a kind of PGM, to pass some intuition. For a complete treatment, see Ref. [1]. Next, I will present more rigorously the theory of Double-Edge Factor Graphs (DEFG) [18], which would be the main framework we will use throughout this work. We choose to work with DEFG and not any other type of PGM because of the strong connection DEFG has with the quantum world, as we will show in the next sections.

3.1 Factor graphs

Consider a multivariate probability distribution $P(x_1, \ldots, x_n)$ over n discrete random variables that can be written in the following 'factorized' form:

$$P(x_1, \ldots, x_n) = \frac{1}{Z} \prod_{\alpha} f_\alpha(\boldsymbol{x}_\alpha). \tag{3.1}$$

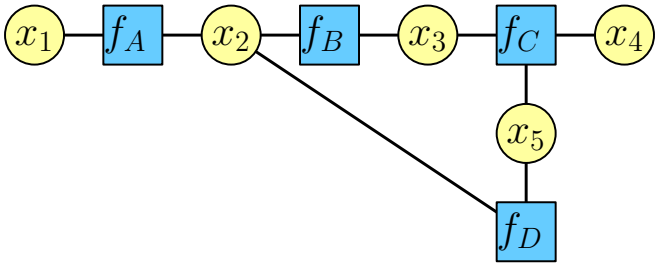
Above, $\boldsymbol{x}_\alpha$ denotes a subset of the variables and the $f_\alpha(\boldsymbol{x}_\alpha)$'s are non-negative functions. The $\frac{1}{Z}$ factor is for an overall normalization and is given by

$$Z = \sum_{x_1,\ldots,x_n} \prod_{\alpha} f_\alpha(\boldsymbol{x}_\alpha). \tag{3.2}$$

Clearly, a great deal of understanding of the statistical dependencies between the different variables in Eq. (3.1) comes from different subsets of variables $\{\boldsymbol{x}_\alpha\}$ and their overlap. A convenient graphical notation that captures this structure is known as a *factor graph* representation. In this notation, every x_i variable is described by a circle and every factor function $f_\alpha(\boldsymbol{x}_\alpha)$ by a square. For example, consider the next joint probability distribution factorization

$$P\left(x_1, x_2, x_3, x_4, x_5\right) = \frac{1}{Z} f_A\left(x_1, x_2\right) f_B\left(x_2, x_3\right) f_C\left(x_3, x_4, x_5\right) f_D\left(x_2, x_5\right) \tag{3.3}$$

Then, the right hand side of the equation above is the factor graph representation of $P\left(x_1, x_2, x_3, x_4, x_5\right)$ and it could be illustrated as follows:



Given some multivariate probability distribution, such as in Eq. (3.1), a common task would be to calculate its marginals over a small number of variables. Such marginals can then be used to calculate averages of local observables. Intuitively, this is a difficult task, as it involves a summation over an exponential number of configurations. Formally, it can be shown to be NP-hard, even if one is interested in a crude approximation for these marginals [23]. The Belief Propagation (BP) algorithm is a heuristic, iterative, message-passing algorithm that tries to do just that. It is exact on factor graphs with a tree structure and often gives surprisingly good results on more general, 'loopy graphs'. To present the algorithm, let us consider the case of a tree graph first, on which it produces the exact result.

3.1.1 The BP equations on tree factor graphs

We start by introducing some notation. For every node index i, denote by N_i the set of indices of factors that are adjacent to it. Similarly, for every factor index α, we denote by N_α the set of node indices that are adjacent to it. Note that in this notation, $\boldsymbol{x}_\alpha = \{x_i | i \in N_\alpha\}$.

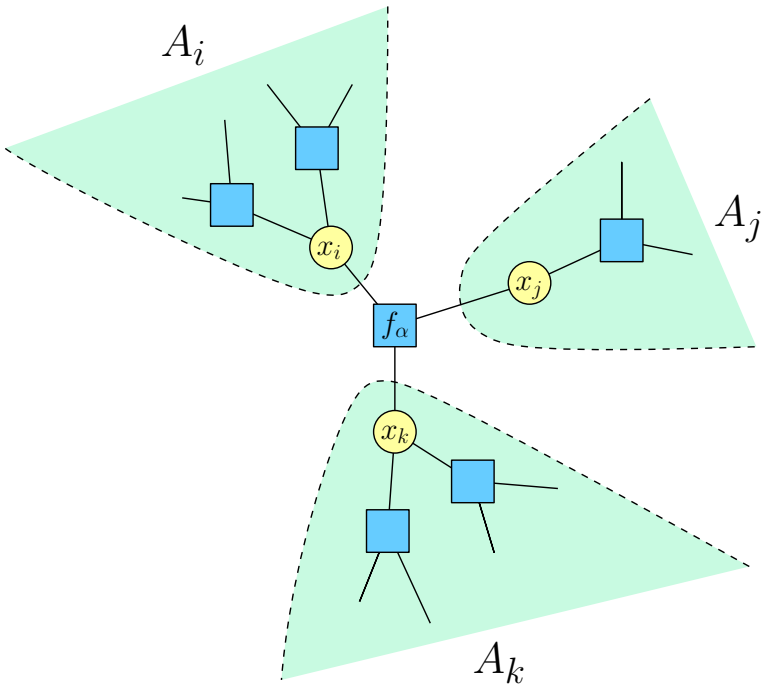
Let A denote a subset of nodes, and define $\boldsymbol{x}_A$ to be the set of variables inside A, and by S_A the set of factors for which *all* their variables are inside A. Finally,

define the "branch function over A" as the function

$$br_A(\boldsymbol{x}_A) \stackrel{\text{def}}{=} \prod_{\alpha \in S_A} f_\alpha(\boldsymbol{x}_\alpha). \tag{3.4}$$

Up to an overall normalization, $br_A(\boldsymbol{x}_A)$ can be viewed as a probability distribution over a trimmed-down system that consists only of the nodes in A and all the factors within.

Let us now assume that we are given a *tree* factor graph, and partition it by breaking all the edges of a factor f_α, as shown in the example below:



Since our graph is a tree, the cut partition it into $|N_\alpha|$ distinct parts: $A_{i_1}, A_{i_2}, \ldots$. Due to the tree structure, we can use the branch functions of the different A_i to write the total probability distribution up to a normalization factor:

$$P(\boldsymbol{x}) \propto f_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in N_\alpha} br_{A_i}(\boldsymbol{x}_{A_i}). \tag{3.5}$$

Moreover, it is easy to see that the *marginal* over $P(\boldsymbol{x}_\alpha)$ is given by

$$P(\boldsymbol{x}_\alpha) \propto f_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in N_\alpha} \sum_{\boldsymbol{x}_{A_i} \setminus \{x_i\}} br_{A_i}(\boldsymbol{x}_{A_i}). \tag{3.6}$$

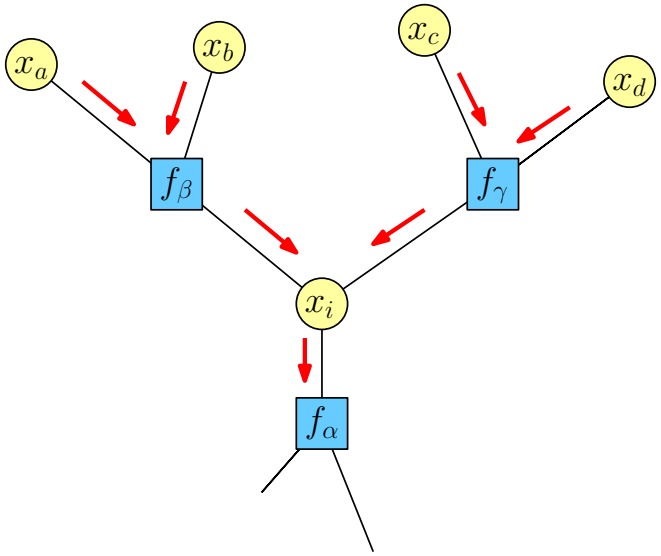
This leads us to define the notion of a *message* $m_{i\to\alpha}(x_i)$ as

$$m_{i\to\alpha}(x_i) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}_{A_i}\setminus\{x_i\}} br_{A_i}(\boldsymbol{x}_{A_i}). \tag{3.7}$$

This can be viewed as the "marginal" of the node i from the trimmed down system that only contains the A_i nodes. Put differently, it can be seen as the what state system A_i "thinks" node x_i should be. With this notation at hand, the marginal over $\boldsymbol{x}_\alpha$ can be compactly written as

$$P(\boldsymbol{x}_\alpha) \propto f_\alpha(\boldsymbol{x}_\alpha) \prod_{i\in N_\alpha} m_{i\to\alpha}(x_i). \tag{3.8}$$

An important observation is that the messages $m_{i\to\alpha}(x_i)$ can be calculated iteratively. To calculate $m_{i\to\alpha}(x_i)$ it is sufficient to know the incoming messages to all the factors that are adjacent to x_i, except for f_α. This is illustrated in the figure below, where $m_{i\to\alpha}(x_i)$ can be calculated from $m_{a\to\beta}(x_a), m_{b\to\beta}(x_b)$ and $m_{c\to\gamma}(x_c), m_{d\to\gamma}(x_d)$:



This relation can be written cleanly if we define the notion of a *message from a factor to a node*:

$$m_{\beta\to i}(x_i) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}_\beta\setminus\{x_i\}} f_\beta(\boldsymbol{x}_\beta) \prod_{j\in N_\beta\setminus\{i\}} m_{j\to\beta}(x_j). \tag{3.9}$$

Then, a *message from a node to factor* would be

$$m_{i\to\alpha}(x_i) = \prod_{\beta\in N_i\setminus\{\alpha\}} m_{\beta\to i}(x_i). \tag{3.10}$$

To summarize, we have reached two mutually consistent equations connecting two types of messages:

$$m_{i\to\alpha}(x_i) = \prod_{\beta\in N_i\setminus\{\alpha\}} m_{\beta\to i}(x_i), \tag{3.11}$$

$$m_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_\alpha\setminus\{x_i\}} f_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in N_\alpha\setminus\{i\}} m_{j\to\alpha}(x_j). \tag{3.12}$$

It is easy to show that starting from the leafs of these tree, where the messages are trivial, we can work our way to the middle of the tree and calculate this way any $m_{i\to\alpha}(x_i)$, without the need to perform the exponential sum over $br_{A_i}(\boldsymbol{x}_{A_i})$ in the original definition 3.7. In fact, it is also easy to show that Eqs. (3.11,3.12) can be solved iteratively, and converge to the right solution after a number of steps that is linear in the system size.

3.1.2 The BP equations on general factor graphs

So far, we have only considered tree factor graphs. The above discussion implied that the problem of calculating marginals on such systems is easy, and can be solved efficiently. What happens when the underlying graph is no longer a tree? In such case, the presence of a loop breaks down many of the properties we used in the above derivation. In fact, it is no longer possible to define $m_{i\to\alpha}(x_i)$ as in (3.7), since breaking up an edge no longer splits the system into two distinguished parts. Related to that, it is well-known that the problem of estimating marginals on a general factor graph is NP-hard, and so we do not expect an efficient algorithm for that to exist.

However, we may still look for a set of messages that solve Eqs. (3.11, 3.12), and use that solution to estimate marginals via Eq. (3.8). Moreover, we can try to solve these equations iteratively by looking for a fix point of

$$m^{(t+1)}_{i\to\alpha}(x_i) = \prod_{\beta\in N_i\setminus\{\alpha\}} m^{(t)}_{\beta\to i}(x_i), \tag{3.13}$$

$$m^{(t+1)}_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_\alpha\setminus\{x_i\}} f_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in N_\alpha\setminus\{i\}} m^{(t)}_{j\to\alpha}(x_j). \tag{3.14}$$

The above equations are known as the BP equations. In practice, they often work extremely well, in particular for systems in which locally look like a tree. Nevertheless, it is crucial to remember that this is essentially a non-controlled approximation, which may (and will) miserably fail sometimes. In addition, still

very little is known on when these iterative equations will converge, and how well their resultant marginals approximate the exact marginals. In cases when the equations do converge one can use the messages to calculate node and factor approximated marginals, also known as beliefs, following the next expressions

$$b_i(x_i) = \prod_{\alpha \in N_i} m_{\alpha \to i}(x_i) \tag{3.15}$$

$$b_\alpha(\boldsymbol{x}_\alpha) = f_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in N_\alpha} m_{i \to \alpha}(x_i). \tag{3.16}$$

3.2 BP solutions and Bethe free energy

As we mentioned, a general understanding on the conditions for convergence of the BP equations, and their quality as approximations is still lacking. Nevertheless, there is an important connection between these solutions and the so-called Bethe free energy, which was explicitly put forward by Yedidia et al in Ref. [1].

To describe this connection, recall that the main problem we are trying to solve is to approximate the local marginals of some factor graph. One approach to solve this problem is to approximate the underlying probability distribution by another, simpler distribution, whose marginals can be easily calculated. For example, we may look to approximate $P(\boldsymbol{x})$ of Eq. (3.1) by a simple product distribution $\prod_i P_i(x_i)$, where $P_i(x_i)$ is a simple univariable probability distribution. In physics, this type of approximation is known as the *mean-field approximation*. More generally, given a family of "simple distributions", we would like to choose the one that is the best approximation to $P(\boldsymbol{x})$. Clearly, the particular approximation we will obtain depends not only on the family of distributions over which we look for an approximation, but also on the figure of merit we use to measure the distance from the approximate distribution to the actual distribution. In information theory, an extremely useful measure for the distance between two probability distributions is the Kullback–Leibler (KL) divergence, which is also known in statistical mechanics as the relative entropy and is given by

$$D_{KL}(P(\boldsymbol{x})||q(\boldsymbol{x})) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}} P(\boldsymbol{x}) \big(\log P(\boldsymbol{x}) - \log q(\boldsymbol{x}) \big). \tag{3.17}$$

Note that the KL divergence is not a proper distance, as it is not symmetric between P, q. Nevertheless, it is always non-negative, and vanishes only when $P = q$.

It turns out that looking for probability distribution that minimizes the KL

divergence is equivalent to a well-known procedure in statistical mechanics. To see this, we first rewrite the factor-graph distribution in Eq. (3.1) as the equilibrium distribution of a fictitious system that is in contact with a heat bath of inverse temperature $\beta = 1$:

$$P(\boldsymbol{x}) = \frac{1}{Z} e^{-\beta H(\boldsymbol{x})} = \frac{1}{Z} e^{-\beta \sum_\alpha h_\alpha(\boldsymbol{x}_\alpha)}. \tag{3.18}$$

The distribution above is known as the *Gibbs distribution* (also as the *Boltzmann distribution*), and in order for it to match the factor graph distribution from Eq. (3.1), we define

$$h_\alpha(\boldsymbol{x}_\alpha) \stackrel{\text{def}}{=} -\ln f_\alpha(\boldsymbol{x}_\alpha). \tag{3.19}$$

The $h_\alpha(\boldsymbol{x}_\alpha)$ can be viewed as fictitious local interactions among the degrees of freedom in $\boldsymbol{x}_\alpha$ and $H(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_\alpha h_\alpha(\boldsymbol{x}_\alpha)$ is the Hamiltonian of the system.

Let us now fix the Hamiltonian $H(\boldsymbol{x}) = \sum_\alpha h_\alpha(\boldsymbol{x}_\alpha)$, and use it to define the following functional with some arbitrary probability distribution $q(\boldsymbol{x})$:

$$F[q] \stackrel{\text{def}}{=} \langle H \rangle_q - TS(q), \tag{3.20}$$

where T denotes the temperature of the system, $\langle H \rangle_q$ is the average energy of the system with respect to $q(\boldsymbol{x})$ and $S(q)$ is the entropy of q such that:

$$\langle H \rangle_q \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}} q(\boldsymbol{x}) H(\boldsymbol{x}), \qquad S(q) \stackrel{\text{def}}{=} -\sum_{\boldsymbol{x}} q(\boldsymbol{x}) \ln q(\boldsymbol{x}). \tag{3.21}$$

In statistical physics, $F[q]$ is called the *Helmholtz free energy* of the system. Physically, for a system in thermal equilibrium at temperature T, it is the amount of work that can be extracted from the system while keeping its volume and temperature fixed. Mathematically, $F[q]$ has the following well-known property [14]:

$$F[q] = F[P] + T \cdot D_{KL}(q||P). \tag{3.22}$$

Therefore, the Gibbs distribution is the one that minimizes the free energy of a given Hamiltonian. Additionally, minimizing the free energy over a set of distributions is equivalent to minimizing the KL divergence with respect to the Gibbs distribution over that set.

Over the years, physicists have invested a lot of efforts in finding techniques

to minimize the free energy. One of these techniques is using the so-called Bethe free energy. In its core it relies on the fact that when the underlying factor graph is a tree, then $P(\boldsymbol{x})$ can be written in terms of its local marginals:

$$P(\boldsymbol{x}) = \frac{\prod_\alpha P_\alpha(\boldsymbol{x}_\alpha)}{\prod_i [P_i(x_i)]^{d_i - 1}}, \tag{3.23}$$

where $P_\alpha(\boldsymbol{x}_\alpha)$ is the marginal over $\boldsymbol{x}_\alpha$, $P_i(x_i)$ is the marginal over the $i'th$ node and d_i indicates the number of factor neighbors the $i'th$ node has. This form of the probability distribution suggests that given a model with a tree structure, we should minimize the free energy over a set of probabilities that are given by

$$q(\boldsymbol{x}) = \frac{\prod_\alpha q_\alpha(\boldsymbol{x}_\alpha)}{\prod_i [q_i(x_i)]^{d_i - 1}}, \tag{3.24}$$

with the obvious consistency conditions between $q_i(x_i)$ and $q_\alpha(\boldsymbol{x}_\alpha)$. In such case, the free energy takes on the following simple form:

$$F_{Bethe}[q] = \sum_\alpha \sum_{\boldsymbol{x}_\alpha} q_\alpha(\boldsymbol{x}_\alpha) \log \frac{q_\alpha(\boldsymbol{x}_\alpha)}{f_\alpha(\boldsymbol{x}_\alpha)} + \sum_i (1 - d_i) \sum_{x_i} q_i(x_i) \log q_i(x_i). \tag{3.25}$$

Minimizing Eq. (3.25) over all distributions of the form in Eq. (3.24) will yield the exact distributions when the underlying graph is a tree. The idea of the Bethe free energy approximation is to minimize Eq. (3.25) over distributions as Eq. (3.24) also when the underlying graph is *not* a tree. We note that in such case:

- The resultant distribution will not necessarily have $q_\alpha(\boldsymbol{x}_\alpha)$, $q_i(\boldsymbol{x}_i)$, as its marginals
- $F[q]$ will not be necessarily equal to $F_{Bethe}[q]$, and consequently, the resultant distribution will not necessarily minimize the true free energy over the set of distributions in Eq. (3.24).

Nevertheless, the experience of over 50 years has shown that this approximation is often very good, especially for systems whose interaction graph looks locally like a tree.

We are now in a position to explain the connection between the BP algorithm and the Bethe free energy approximation. As shown by Yedidia et al. in Ref. [1], the fixed points of the BP equations correspond to the extremum points of the Bethe free energy. Specifically, the beliefs that we get from the converged BP equations are extremum points of $F_{Bethe}[q]$. The fact that the optimum points

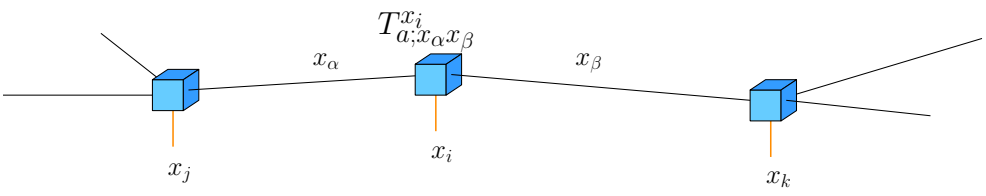
of the Bethe free energy often give good approximations to the true free energy of the system explains why the BP algorithm often performs well.

3.3 Double-Edge Factor Graphs (DEFG)

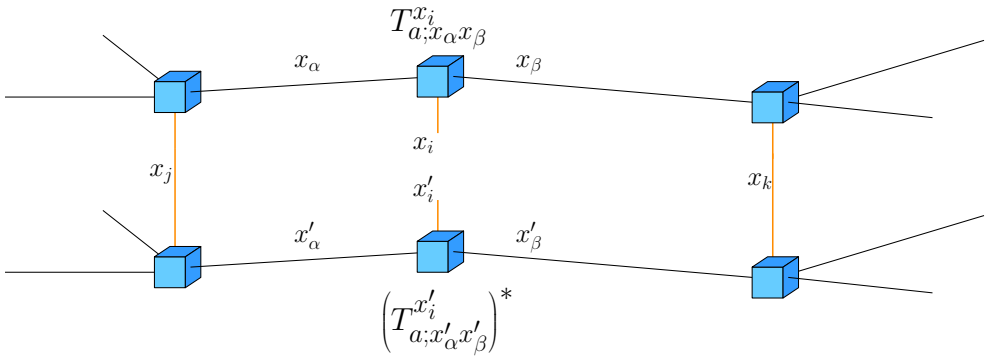
There are many lines of similarity between many-body quantum states and multivariate probability distributions. First, both take an exponential amount of parameters to specify. Second, graphical models, like tensor networks, aim to model a small, yet a physically relevant fraction of states using a polynomial family of parameters. Moreover, inspite of their saving in space, in both cases, it might still be challenging to use these representations to calculate marginals (or reduced density matrices, in the quantum case) or expectation values of local observables. These similarities suggest that it might be possible to use the framework of graphical models to represent quantum states — much like tensor-networks do. Moreover, such representation might enable us to import techniques from the world of graphical models, such as the BP algorithm, to the quantum world.

On the other hand, there are essential differences between probability distributions and quantum states, making such mapping not completely trivial. Quantum states, or more precisely, their expansion in a given basis, are described by complex numbers. On the other hand, ordinary graphical models are defined by real and positive functions $f_\alpha(\boldsymbol{x_\alpha})$. Therefore, to describe quantum states, we will introduce a new kind of PGM called Double-Edge Factor Graphs (DEFG) that can handle complex factors, and was first introduced in Ref. [18]. In the rest of this section, we will explain this framework and how it can be used to describe quantum states. However, first, I would like to open with an example that will emphasize the problem of using factor graphs in the quantum framework.

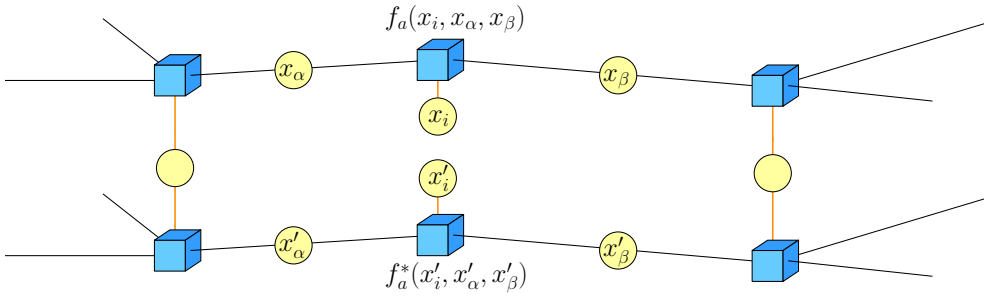
Example: Consider an N spins quantum state tree tensor network representation $|\Psi\rangle_{\text{Tree}}$. Let us zoom in on the i'th spin/tensor



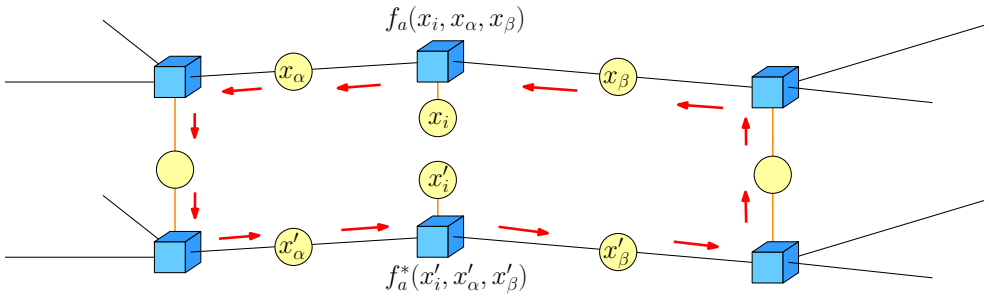
We can compute the i'th spin reduced density matrix ρ_i by sandwiching the tensor network with its complex conjugate through the physical edges (in orange) while keeping the x_i, x_i' edges open as follows:



In order to find ρ_i, one needs to contract all the non-open edges in the tree tensor network above. Another way could be by exploiting the factor graph and BP algorithm from the previous section. There are several ways in which we can map the problem into the language of graphical models. Probably the most natural way is as follows: First, we introduce a node for every edge and a factor for every tensor. The tree tensor network representing ρ_i transforms as follows:



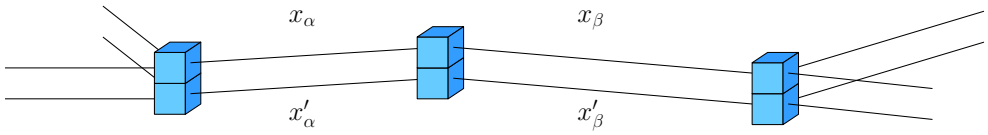
It is now easy to see that the marginal over x_i, x'_i, which by definition is the summation over all other variables in the graphical model, is exactly the reduced density matrix over the i'th particle. To approximate it, we may try the BP algorithm, and use the converged messages to calculate the marginal over x_i, x'_i. There are two problems in this idea: First, although $|\Psi\rangle_{\text{Tree}}$ is a tree, $\rho_i = \text{Tr}_{\boldsymbol{x}\setminus x_i, x'_i}(|\Psi\rangle_{\text{Tree}}\langle\Psi|_{\text{Tree}})$ is not, its factor graph has loops



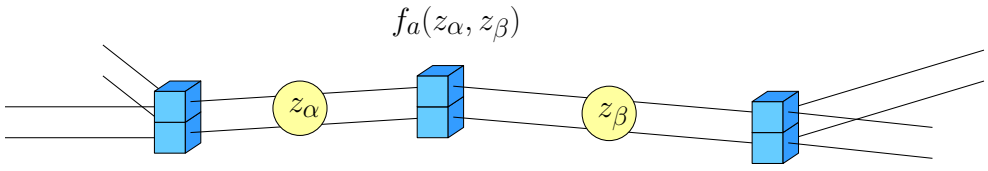
Therefore, the BP algorithm might not converge, and even if it does, it may give us a bad approximation, whereas the contraction of a tree-TN can be done exactly in polynomial time. Moreover, the resultant approximation might suffer

from two crucial problems. First, from the BP rules of calculating marginals we get that the resultant density matrix would be of a product form $\rho_i = b_i(x_i)b_{i'}(x'_i)$ — which could only happen if the i'th particle is unentangled with the rest of the system (which is almost never the case). Second, unless $b_i(x) = \big(b_{i'}(x)\big)^*$ for every x, the resultant density matrix would not be positive semi-definite, and hence not representing a legal quantum state.

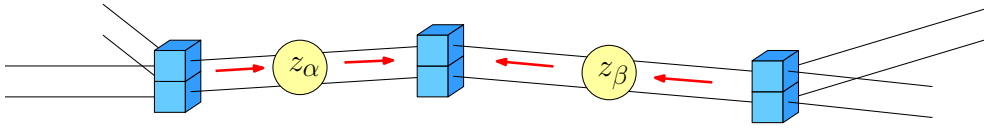
To overcome these problems, we recall that the BP messages were defined on a tree-graph in a way that they are equal to the exact marginals of the trimmed sub-graphs (see Eq. (3.7)). Therefore, if we could turn the graph above into a tree, we are guaranteed that the messages would converge to their true values, which will get us the exact marginals. For that we turn back to the tensor network we had at the beginning and contract all the physical edges (including the open ones) as follows:



Then, we transform the tensor network to a graphical model as follows: every pair of matching virtual edges x_α, x'_α are introduced with a single node. Also, every pair of tensors T_a, T_a^* are introduced with a factor which equals the contraction of the physical edge between the two. Thus, we end up with the next factor graph

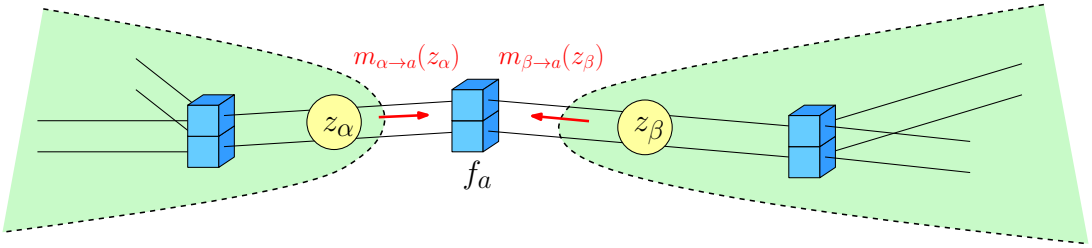


where $f_a(z_\alpha, z_\beta) = \mathrm{Tr}_{x_i}\left[\left(T^{x_i}_{a;x_\alpha,x_\beta}\right)\left(T^{x_i}_{a;x'_\alpha,x'_\beta}\right)^*\right]$, and for convenience we define $z_\alpha \overset{\text{def}}{=} (x_\alpha, x'_\alpha)$ and $z_\beta \overset{\text{def}}{=} (x_\beta, x'_\beta)$. Next, we preform BP on double-edges which we will define rigorously later. For now, it means that messages are sent on double-edges instead on single ones as in the illustration below

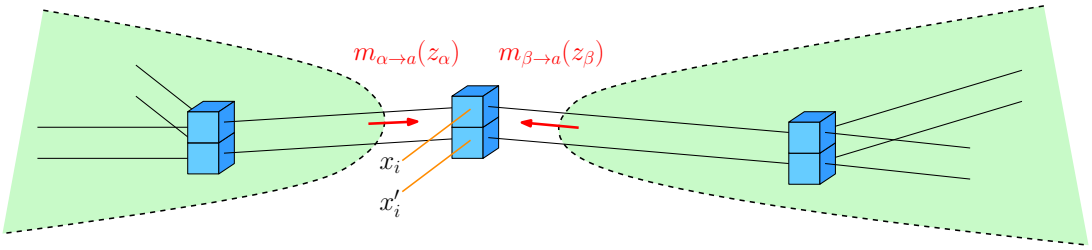


This implies that the messages are now matrices instead of vectors as before. A nice property of BP on double-edges which we will prove later is that if we initialize the messages as positive semi-definite matrices, BP on double-edges will

preserve that property throughout the iterative process. In addition, notice that the resultant graph is also a tree (with respect to the BP process). Thus, the BP algorithm is guaranteed to converge to the exact marginals. We use BP just to get the incoming messages from the sides of f_a



Then, once we get them, we "break" the fused factor and go back to the TN picture to calculate the marginals properly.



This way, BP is merely an instrument to calculate the contraction over the branches of the tree. The $b_i(x_i, x_i')$ marginal could be computed following the illustration above as follows:

$$b_i(x_i, x_i') = \frac{1}{Z_i} \sum_{z_\alpha, z_\beta} f_a(x_i, x_i', z_\alpha, z_\beta) m_{\alpha\to a}(z_\alpha) m_{\beta\to a}(z_\beta) = \rho_i \tag{3.26}$$

where $f_a(x_i, x_i', z_\alpha, z_\beta) = \left(T^{x_i}_{a;x_\alpha,x_\beta}\right)\left(T^{x_i'}_{a;x_\alpha',x_\beta'}\right)^*$, Z_i is the normalization factor and the two messages are the ones coming from the trimmed branches on both sides. A general form of the equation above would be

$$b_i(x_i, x_i') = \frac{1}{Z_i} \sum_{\boldsymbol{z}_a} f_a(x_i, x_i', \boldsymbol{z}_a) \prod_{j\in N_a} m_{j\to a}(z_j) = \rho_i \tag{3.27}$$

where $\prod_{j\in N_a} m_{j\to a}(z_j)$ indicates the messages coming from all branches into f_a. Thus, we see that if the graph is a tree, then the double-edge $b_i(x_i, x_i')$ belief is equal to the corresponding reduced density matrix. That result justifies the whole usage of BP on double-edges. Moreover, as for BP on loopy single edges, it turns out that even on loopy double-edge we can still use Eq. (3.27) to calculate the reduced density matrix, but then it would become an approximation rather than exact.

Next, we formalize the above discussion by introducing the notion of Double-Edge Factor Graphs (DEFG) and PSD functions.

Definition 3.3.1 (PSD function)
Let Σ be a finite alphabet. Then a function $f : \Sigma \times \Sigma \to \mathbb{C}$ is said to be Positive semi-definite (PSD) if for every complex-valued function $h : \Sigma \to \mathbb{C}$ it holds that

$$\sum_{\boldsymbol{x},\boldsymbol{x}'} h(\boldsymbol{x}) f(\boldsymbol{x},\boldsymbol{x}') h^*(\boldsymbol{x}') \geq 0 \tag{3.28}$$

where h^* denotes the complex conjugate of h.

Remark: if we think of $f(\boldsymbol{x},\boldsymbol{x}')$ as a matrix and $h(\boldsymbol{x}), h^*(\boldsymbol{x}')$ as vectors, then the definition coincides with the common definition for PSD matrices.

Next, let us define properly a double-edge factor graph.

Definition 3.3.2 (DEFG)
A Double-Edge Factor Graph (DEFG) is a bipartite graph $\mathcal{G}(\mathcal{V},\mathcal{F},\mathcal{E})$ representing the factorization of a PSD function $g(\boldsymbol{x},\boldsymbol{x}')$:

$$g(\boldsymbol{x},\boldsymbol{x}') = \prod_{a=1}^{N} f_a(\boldsymbol{x}_a,\boldsymbol{x}'_a) \tag{3.29}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_S)$, $\boldsymbol{x}' = (x'_1, x'_2, ..., x'_S)$ are discrete variables over a finite alphabet, and $\boldsymbol{x}_a, \boldsymbol{x}'_a$ are subsets of $\boldsymbol{x}, \boldsymbol{x}'$ respectively. For mathematical convenience we define $z_i \stackrel{\text{def}}{=} (x_i, x'_i)$ for every i such that the factorization above takes the next compact form

$$g(\boldsymbol{z}) = \prod_{a=1}^{N} f_a(\boldsymbol{z}_a) \tag{3.30}$$

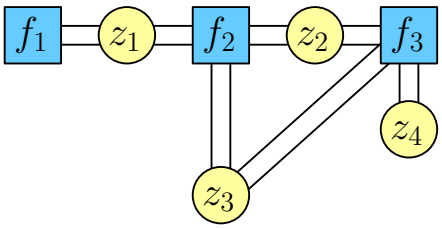
where $\boldsymbol{z}_a = (\boldsymbol{x}_a, \boldsymbol{x}'_a)$. The DEFG, $\mathcal{G}(\mathcal{V},\mathcal{F},\mathcal{E})$, consists of three sets:

- $\mathcal{V}$: *The set of nodes $\mathcal{V} = \{z_1, z_2, ..., z_S\}$.*
- $\mathcal{F}$: *The set of factors $\mathcal{F} = \{f_1, f_2, ..., f_N\}$, s.t. every $f_a(\boldsymbol{z}_a)$ is a PSD function operating from the alphabet space of $\boldsymbol{z}_a = (\boldsymbol{x}_a, \boldsymbol{x}'_a)$ into $\mathbb{C}$.*
- $\mathcal{E}$: *The set of all undirected double-edges. A double-edge is connecting node z_i and a factor f_a iff $i \in N_a$ where N_a denote the set of all neighboring nodes of factor f_a.*

Let us see an example. Given the function $g(z_1, z_2, z_3, z_4)$ with the next factorization

$$g(z_1, z_2, z_3, z_4) = f_1(z_1) f_2(z_1, z_2, z_3) f_3(z_2, z_3, z_4) \tag{3.31}$$

its DEFG representation is given by:



where notice that when denoting every pair (x_i, x_i') with a z_i, the graph looks exactly like a regular factor graph up to the fact that it has double-edges instead of single ones. This is not the only similarity between the two graphical models. In DEFG, as in FG, we also would like to calculate local marginals, which is known to be a difficult task. Therefore, we will use the BP algorithm in order to evaluate the approximated marginals. The BP algorithm on DEFG is essentially the same as on FG up to few main differences:

1. In the DEFG framework, the BP messages are PSD matrices, rather then vectors, as in the FG framework. We will prove this property shortly.

2. Being complex-valued functions, the messages, beliefs and marginals loose their classical probabilistic interpretation. However, because they are PSD, we will be able to interpret them as density matrices.

It is important to note that the reason we insist on sending the BP messages on double-edges is to preserve positivity at any step of the calculation (will be proven later), which is an essential property of density matrices. Otherwise, DEFG are just graphical models with complex variables.

As mentioned, we can us the BP algorithm over the DEFG to approximate its marginals. Using the $z_i \stackrel{\text{def}}{=} (x_i, x_i')$ notation, the BP equations in DEFG consists of two kinds of messages: factor-to-node and node-to-factor and look exactly like the ones on FG. Also here, the BP messages corresponds to trimmed subgraphs of the tree-DEFG. In case where loops introduced in the DEFG, both kinds of messages would become an approximation. The approximation would

be affected by the amount and size of loops in the given DEFG. Given the two kinds of messages, the BP equations are as follows:

$$m_{a\to i}^{(t+1)}\left(z_i\right) \propto \sum_{\boldsymbol{z}_a \backslash z_i} f_a\left(\boldsymbol{z}_a\right) \prod_{j \in N_a \backslash i} m_{j\to a}^{(t)}\left(z_j\right) \tag{3.32}$$

$$m_{i\to a}^{(t+1)}\left(z_i\right) \propto \prod_{b \in N_i \backslash a} m_{b\to i}^{(t)}\left(z_i\right) \tag{3.33}$$

where at $t=0$ all messages $m_{a\to i}^{(0)}\left(z_i\right)$, $m_{i\to a}^{(0)}\left(z_i\right)$ are initialized as PSD matrices. The most common update scheme is the parallel one, where at each step all nodes send messages to their neighboring factors and after all factors are updated they send messages to all of their neighboring nodes. The iterative process convergence criterion is usually taken with respect to the difference between messages at consecutive steps of the algorithm as $|m_{a\to i}^{(t+1)} - m_{a\to i}^{(t)}| < \epsilon$, for all the messages, where ϵ is some predetermined constant.

As we mentioned above, a significant difference between BP in FG on single-edges and BP in DEFG on double-edges is that the second preserves the positive semi-definiteness of the messages throughout the iterative process. In order to prove that statement we will use the next theorem.

Theorem 3.3.3 (Schur's product theorem)
The product of any two PSD functions is also a PSD function.

Proof: Given two PSD functions f and g which operate on the same dimension $(\boldsymbol{x}, \boldsymbol{x}')$ and some complex-valued function $h \neq 0$. Then, both f and g can be decomposed as follows:

$$f\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \sum_{K_f} b_{K_f}\left(\boldsymbol{x}\right) b_{K_f}^{*}\left(\boldsymbol{x}'\right) \tag{3.34}$$

$$g\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \sum_{K_g} b_{K_g}\left(\boldsymbol{x}\right) b_{K_g}^{*}\left(\boldsymbol{x}'\right) \tag{3.35}$$

then,

$$\sum_{\boldsymbol{x},\boldsymbol{x}'} h\left(\boldsymbol{x}\right) f\left(\boldsymbol{x}, \boldsymbol{x}'\right) g\left(\boldsymbol{x}, \boldsymbol{x}'\right) h^{*}\left(\boldsymbol{x}'\right) = \sum_{\boldsymbol{x},\boldsymbol{x}'} h\left(\boldsymbol{x}\right) \left(\sum_{K_f} b_{K_f}\left(\boldsymbol{x}\right) b_{K_f}^{*}\left(\boldsymbol{x}'\right) \right) \tag{3.36}$$

$$\left(\sum_{K_g} b_{K_g}\left(\boldsymbol{x}\right) b_{K_g}^{*}\left(\boldsymbol{x}'\right) \right) h^{*}\left(\boldsymbol{x}'\right) \tag{3.37}$$

$$= \sum_{K_f, K_g} \sum_{\boldsymbol{x},\boldsymbol{x}'} h\left(\boldsymbol{x}\right) b_{K_f}\left(\boldsymbol{x}\right) b_{K_g}\left(\boldsymbol{x}\right) \tag{3.38}$$

$$h^{*}\left(\boldsymbol{x}'\right) b_{K_f}^{*}\left(\boldsymbol{x}'\right) b_{K_g}^{*}\left(\boldsymbol{x}'\right) \tag{3.39}$$

$$= \sum_{K_f,K_g} \left(\sum_{\boldsymbol{x}} h\left(\boldsymbol{x}\right) b_{K_f}\left(\boldsymbol{x}\right) b_{K_g}\left(\boldsymbol{x}\right) \right) \tag{3.40}$$

$$\left(\sum_{\boldsymbol{x}'} h\left(\boldsymbol{x}'\right) b_{K_f}\left(\boldsymbol{x}'\right) b_{K_g}\left(\boldsymbol{x}'\right) \right)^{*} \tag{3.41}$$

$$= \sum_{K_f,K_g} C_{K_f,K_g} C_{K_f,K_g}^{*} \tag{3.42}$$

$$\geq 0 \tag{3.43}$$

■

We will use Schur's product theorem in order to prove the next central proposition which is the heart of the BP algorithm on DEFG.

Proposition 3.3.4 (BP preserves PSD of messages)
Assume that at $t = 0$ all the BP double-edge messages initialized as PSD matrices. Then, for every BP iteration $t \geq 1$ the double-edge messages are kept complex-valued PSD matrices.

Proof: In Eqs. (3.33, 3.32), we see that the messages at $t+1$ are product of the messages at t and some factor functions. Since all of them are PSD, by Schur's product theorem 3.3.3 also the $t+1$ messages would be PSD.

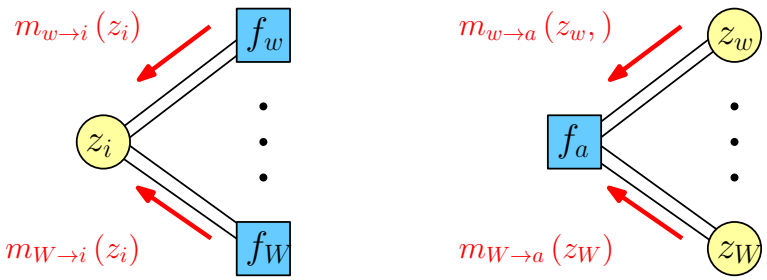
■

Now, running BP on some DEFG, after the messages convergence criterion is met, the messages can be used for computing the set of node and factor 'beliefs' (see Fig. 3.1) as follows:

$$b_i\left(z_i\right) \propto \prod_{a \in N_i} m_{a \to i}\left(z_i\right) \tag{3.44}$$

$$b_a\left(\boldsymbol{z}_a\right) \propto f_a(\boldsymbol{z}_a) \prod_{i \in N_a} m_{i \to a}\left(z_i\right) \tag{3.45}$$

The reason we insist on using the name 'beliefs' in that context is to remind ourselves the connection to the beliefs in the FG framework which computed in the same way. Different from the FG beliefs, the DEFG beliefs are PSD complex-valued functions. Therefore, cannot be interpreted in any way as local classical probabilities.



(a) (b)

Figure 3.1: (a) Node belief illustration. (b) Factor belief illustration.

Similarly to BP in FG, if the DEFG has no loops, the beliefs would be exactly equal to their associated marginals (see sec. 3.1.2). In cases where loops are introduced, they become an uncontrolled approximation of the marginals which depends on the size and number of loops in the graph. As we will see empirically in Chapter 5, as in the FG case, even in loopy DEFG the BP process often gives good results.

3.3.1 TN — DEFG transformation and reduced density matrices in DEFG

In this section, we show how to use the BP messages in DEFG to calculate quantum state reduced density matrices. Before diving in, let us properly define the TN — DEFG transformation. Given a sandwiched TN (i.e. $\langle\Psi|\Psi\rangle$), one could construct its associated DEFG following the next two steps:

1. Every pair of conjugated tensors $T^{\mu}_{a;x_1x_2\ldots x_k}, \left(T^{\mu'}_{a;x'_1x'_2\ldots x'_k}\right)^*$ are introduced with a factor as follows:

$$f_a[\boldsymbol{z}_a] = f_a[z_1, z_2, ..., z_k] \tag{3.46}$$
$$= f_a[(x_1, x'_1), (x_2, x'_2), ..., (x_k, x'_k)] \tag{3.47}$$
$$= \sum_{\mu=1}^{d} \left(T^{\mu}_{a;x_1x_2\ldots x_k}\right) \cdot \left(T^{\mu}_{a';x'_1x'_2\ldots x'_k}\right)^* \tag{3.48}$$

   where we denote the physical leg with μ.

2. Every pair of adjacent edges x_i, x'_i are introduced with a double-edge node z_i.

Example of the TN — DEFG transformation for a rank-3 tensor is illustrated in Fig. 3.2.

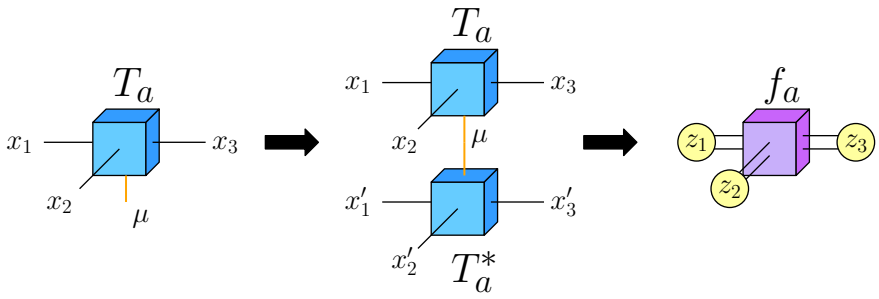


Figure 3.2: Illustrations of the TN — DEFG transformation for a rank-3 tensor.

Now, consider a DEFG resulted from some TN following this transformation. We can calculate 1-body reduced density matrices following the next equation:

$$\rho_i = \frac{1}{Z_i} \sum_{\boldsymbol{z}_a} f_a(x_i, x_i', \boldsymbol{z}_a) \prod_{j \in N_a} m_{j \to a}(z_j) \tag{3.49}$$

where $m_{j \to a}(z_j)$ are the converged double-edge BP messages entering f_a from its neighboring nodes. Notice that Eq. (3.49) end up to be very similar to the *factor belief* given in Eq. (3.45). The difference is that in the reduced density matrix calculation the physical variables are left uncontracted. Therefore, we replace the factor from Eq. (3.45) with an uncontracted factor given by

$$f_a(x_i, x_i', \boldsymbol{z}_a) \overset{\text{def}}{=} \left(T^{x_i}_{a;x_1 x_2 \ldots x_k}\right) \cdot \left(T^{x_i}_{a';x_1' x_2' \ldots x_k'}\right)^* \tag{3.50}$$

Now, Eq. (3.49) could be easily scaled to k-body reduced density matrices calculation by multiplying the k relevant factors and their entering messages all together. In a tree DEFG, this calculation would give the exact reduced density matrices. If the TN (therefore also the DEFG) has loops in it, we can still use Eq. (3.49) by simply ignoring them. The resultant reduced density matrices would then be an approximation to the real ones. This approximation would be dependent on the amount of loops, size of loops and the entanglement between the physical degrees of freedom.

Chapter 4

Algorithms for finding ground states of spins systems

We have seen in chapter 2 that TN representations are useful tools for approximating many-body quantum states and calculating their local observables expectation values. We also have seen that the MPS and PEPS representations satisfy the entanglement entropy area law, which is a property that is believed to appear in ground states of gapped local Hamiltonians. Therefore, there is a high incentive to use the MPS and PEPS representations to efficiently approximate those kinds of quantum states. For that purpose alone, many TN algorithms have been developed, such as Full Update (FU), Fast Full Update (FFU), Loop Update (LU), Simple Update (SU), etc [22][6][5]. The one thing that all those algorithms have in common is that they are all trying to approximate TN local environments and use them to evolve the quantum state in time. In this chapter, I will first elaborate on the Imaginary Time Evolution (ITE) method, which is used for evolving quantum states and finding ground states of local Hamiltonians, and present some of its limitations. After that, I will specify in detail the Simple Update algorithm and show its connection to ITE. Then, I will introduce the Belief Propagation Update (BPU) algorithm, a new ITE based algorithm that we developed. The BPU algorithm is different from all other TN algorithms, as it is based on the BP algorithm to approximate the local environments of the underlying TN.

4.1 Imaginary Time Evolution (ITE)

Consider some local Hamiltonian $\mathcal{H}$. Its ground state $|\Psi_{GS}\rangle$ can be obtained by evolving a randomly initialized state $|\Psi_0\rangle$ (which has a non-vanishing overlap

with $|\Psi_{GS}\rangle$) in imaginary time τ such that

$$|\Psi_{GS}\rangle = \lim_{\tau\to\infty} \frac{\exp(-\tau\mathcal{H})|\Psi_0\rangle}{\|\exp(-\tau\mathcal{H})|\Psi_0\rangle\|} \tag{4.1}$$

where $U(\tau) = \exp(-\tau\mathcal{H})$ is the so called ITE operator. In order to use that method in the TN framework, we need to split this global operator into local terms. Then, those terms could be used to evolve local parts in a given TN in imaginary time to approximate the ground state of $\mathcal{H}$. In the case of nearest-neighbors interactions, $\mathcal{H}$ can be written as a sum of local terms $\mathcal{H} = \sum_{\langle i,j\rangle} h_{ij}$. Then, for small enough time intervals $\delta\tau = \tau/N$ where $N \gg 1$ we can use the so called first order Trotter-Suzuki decomposition $e^{\delta(A+B)} = e^{\delta A}\cdot e^{\delta B} + O(\delta^2)$ [24], in order to split the ITE operator into local terms as follows:

$$\begin{aligned} U(\tau) &= \exp(-\tau\mathcal{H}) && (4.2)\\ &= [\exp(-\delta\tau\mathcal{H})]^N && (4.3)\\ &= \left[\exp\left(-\delta\tau\sum_{\langle i,j\rangle} h_{ij}\right)\right]^N && (4.4)\\ &= \prod_N \prod_{\langle i,j\rangle} \exp(-\delta\tau h_{ij}) + \mathcal{O}\left((\delta\tau)^2\right) && (4.5) \end{aligned}$$

Then, we identify the two-body local gate approximation as

$$U_{ij}(\delta\tau) = \exp(-\delta\tau h_{ij}) \tag{4.6}$$

The local ITE gate $U_{ij}(\delta\tau)$ is a function of two sites i, j that share a common term h_{ij} in the Hamiltonian. The ITE method is commonly used in the framework of TN, where it is implemented following the next two steps:

Evolution: Given a TN representation $|\Psi\rangle_{TN}$ with bond dimension D, apply the local ITE gate $U_{ij}(\delta\tau)$ over the two i, j interacting sites such that $|\Psi'\rangle_{TN} = U_{ij}(\delta\tau)|\Psi\rangle_{TN}$. Then, the bond dimension of the virtual edge connecting i, j in the new state $|\Psi'\rangle_{TN}$ would be $D' \geq D$, see Fig. 4.1(c).

Truncation: Approximate $|\Psi'\rangle_{TN}$ with a new TN representation, $|\Psi''\rangle_{TN}$, with bond dimension D by truncating the i, j common bond dimension from D' back to D.

The evolution part could be carried out in few forms. The two most common ones are the sequential and parallel, which illustrated for the case of a finite MPS

with open boundary conditions in Fig. 4.1.

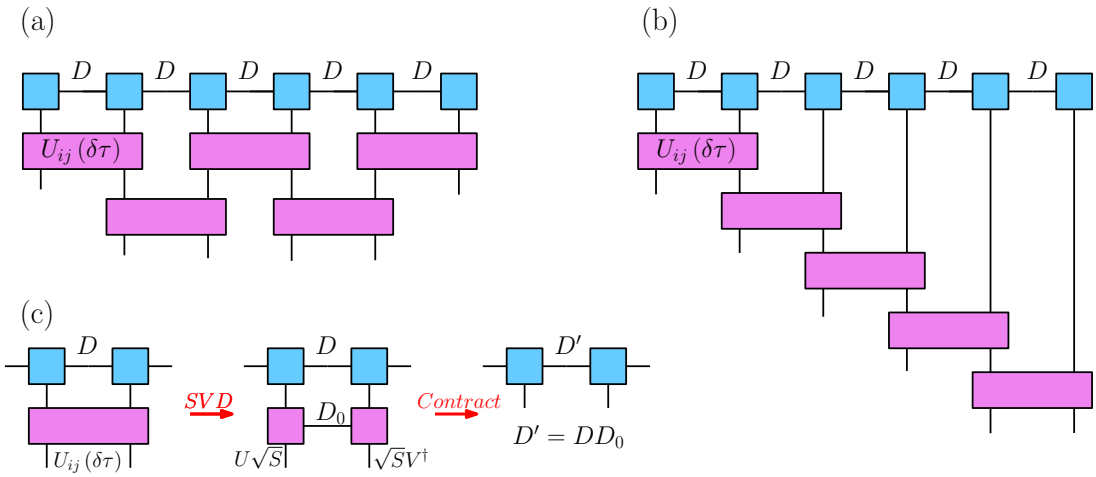
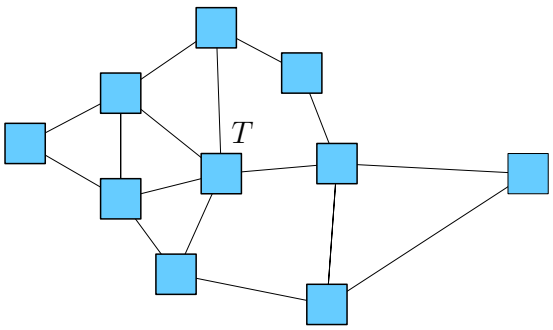
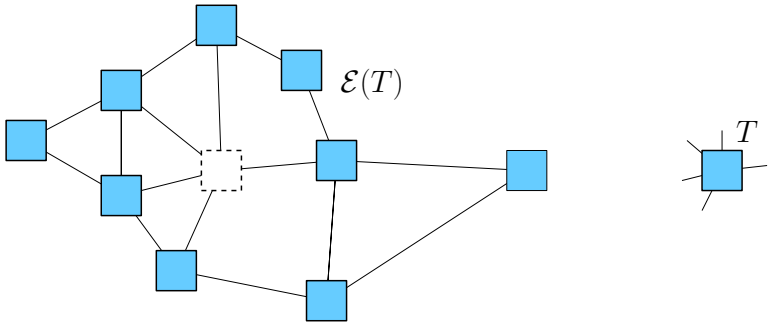


Figure 4.1: Examples of a single $\delta\tau$ evolution step ITE implementation over a finite MPS with bond dimension D. (a) Parallel ITE implementation. (b) Sequential ITE implementation. Notice that in both methods every TN edge is evolved only once. (c) Explanatory illustration of bond dimension growth in evolution step. Notice that $D' = DD_0$, where D_0 is the number of non zero singular values of the local ITE operator.

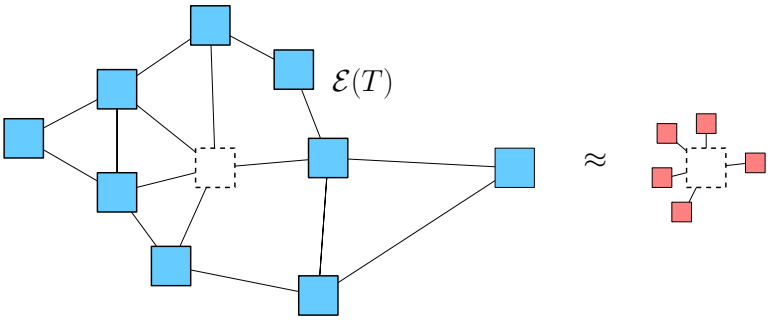
Before elaborating on the truncation step, let us first define the notion of *environment* in TN. Consider the next arbitrary TN representation



The *environment* $\mathcal{E}(T)$ of the tensor T is defined as the TN consisting of all the tensors in the original TN except tensor T as follows:



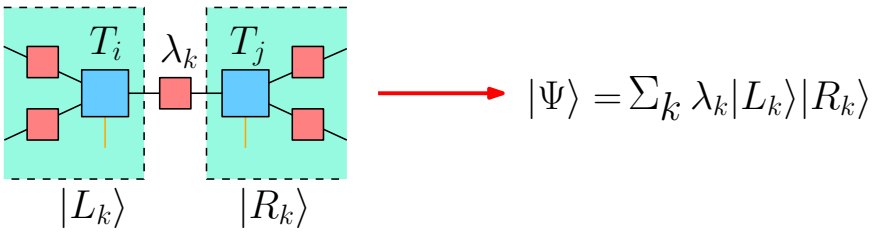
Now back to the ITE truncation step, to perform an optimal truncation, we need to take into account information from the rest of the TN. The quality of the environment approximation has a tremendous effect on the truncation's accuracy. Different TN algorithms use different ways to approximate the environment of a given tensor T. The two extremes are the Full Update (FU) and Simple Update (SU) methods. The FU method takes into account the tensor T full environment $\mathcal{E}(T)$ in the update procedure. It considered as the most accurate and least efficient method. On the other hand, the SU method, which will be introduced in more detail in the next section, approximates the tensor T environment using local diagonal matrices as follows:



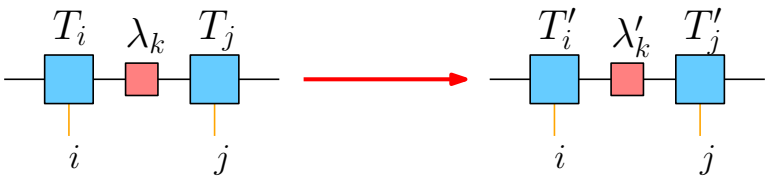
This method, in some sense, is the "zeroth order" TN evolution method where only local information is taken into account in the update procedure. This kind of approximation is highly dependent on the bond dimension magnitude of the local diagonal matrices which determines the number of parameters used in the environment calculation. In some sense, taking the bond dimension to be very large could in theory represent accurately any kind of TN environments. The SU method is the most efficient and least accurate TN evolution method. In the last paragraph, we defined the notion of environment and said it is taking part in the truncation step. In the next section, I will thoroughly explain how this truncation step takes place as part of the SU algorithm.

4.2 Simple Update (SU)

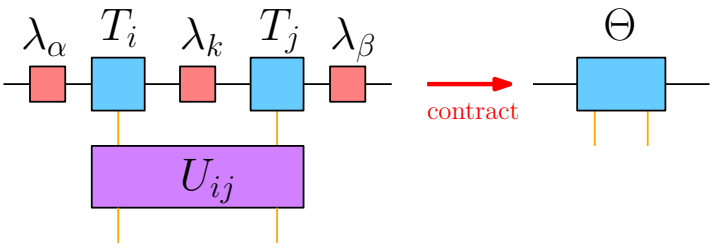
Simple Update (SU) is an ITE based method for finding TN approximations of local Hamiltonians ground-states [25]. It was first introduced on tree-TN, where the environments could be calculated exactly and efficiently using local SVD operations by introducing a λ weight tensor over every virtual edge. This λ tensor corresponds to the Schmidt values resultant form the partitioning of the TN over that edge. Later it was expended to loopy TN, where it uses a rough local approximation of the environments but surprisingly gives decent results in approximating a wide range of quantum states. Also, in many cases, it is used as a seeding step for more complicated and less efficient TN algorithms. In principle, for every tree TN, there exists a unique canonical form that corresponds closely to the Schmidt decomposition of the state at every edge (which breaks the system into two).



One way to obtain the canonical representation of a tree-TN is by iteratively changing every pair of interacting tensors using local SVD operation.

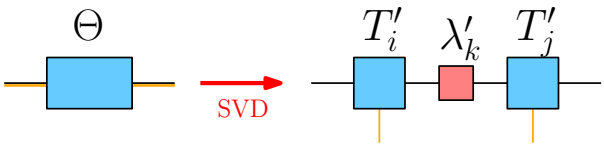


An important property of this process is that the contraction of T_i, λ_k, T_j is equal to the contraction of T'_i, λ'_k, T'_j, therefore the overall TN represents the same state $|\Psi\rangle$. Moreover, if we start with a canonical form, then we can easily keep it while evolving the system: after applying a local ITE gate U_{ij}, we can return to the canonical form using only local SVD operations. For example: consider the two tensors T_i, T_j neighbors of λ_k. Evolving them in imaginary time by contracting them with the local ITE gate U_{ij}, and their environment $\lambda_\alpha, \lambda_\beta$, we end up with a single tensor Θ.



Then, by applying an SVD operation such that $USV^\dagger = \Theta$, we can go back to the canonical form by identifying

$$T'_i \equiv U, \quad \lambda'_k \equiv S, \quad T'_j \equiv V^\dagger. \tag{4.7}$$



Once we obtain the canonical form of a tree-TN, the single and double site reduced density matrices (RDMs) ρ_i, ρ_{ij} are given following the two contraction schemes depicted in Fig. 4.2.

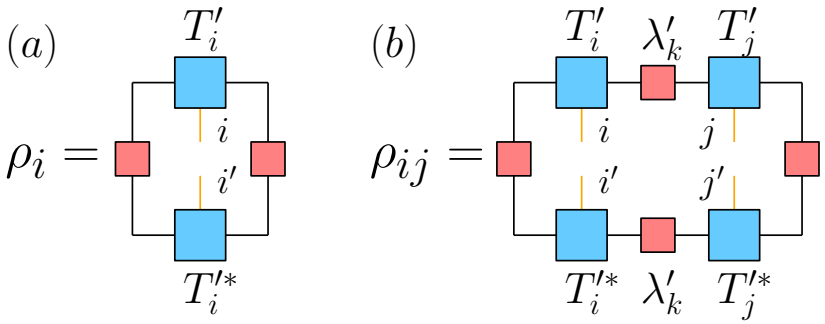


Figure 4.2: Calculating RDMs of tree-TN in canonical form. (a) 1-body RDM ρ_i. (b) 2-body RDM ρ_{ij}.

Although the SU algorithm is based on the tree-TN canonical form, we still use it in the general case of loopy-TN. When the TN contains loops, there is not a unique canonical form, and it is impossible to associate such forms with an underlying Schmidt decomposition. Nevertheless, if we perform a series of SU iterations and converge, we can still use the scheme for ρ_i, ρ_{ij} from Fig. 4.2, which is exact in the case of trees, to get an estimate for the actual RDMs and empirically gives decent results. The complete diagram of the SU algorithm is illustrated in Fig. 4.3 and fully described in Algorithm 4.2.1

Algorithm 4.2.1 (Simple Update (SU))

(a) Pick an edge k and find its two corresponding tensors T_i, T_j.

(b) Absorb bond matrices λ_m to all virtual edges $m \neq k$ of T_i, T_j.

(c) Group all virtual edges $m \neq k$ to form P_l, P_r MPS tensors.

(d) Implement SVD over P_l, P_r to obtain $U_l, S_l, V_l^\dagger$ and $U_r, S_r, V_r^\dagger$

(e) Absorb diagonal matrices to form $R = S_l V_l^\dagger$ and $L = U_r S_r$

(f) Contract the ITE gate U_{ij} with R, L and λ_k to form a Θ tensor.

(g) Preform SVD to Θ in order to obtain $\bar{R}, \bar{\lambda_k}, \bar{L}$ tensors.

(h) Truncate $\bar{R}, \bar{\lambda_k}, \bar{L}$ tensors by keeping only the D largest singular values to obtain $\tilde{R}, \tilde{\lambda_k}, \tilde{L}$.

(i) Glue back the $\tilde{R}, \tilde{L}$ sub-tensors to $U_l, V_r^\dagger$ respectively to form updated tensors $P_l^{'}, P_r^{'}$.

(j) Reshape $P_l^{'}, P_r^{'}$ back to the original tensors T_i, T_j shape to obtain the updated tensors $T_i^{'}, T_j^{'}$.

(k) Remove bond matrices λ_m from all virtual edges $m \neq k$ to obtain the updated tensors $\tilde{T_i}, \tilde{T_j}$

The algorithm is performed iteratively over all TN edges until the convergence criterion is met. The convergence criterion that usually used is the upper bound on the λ weights temporal difference $\|\lambda_k^{(t+1)} - \lambda_k^{(t)}\|_2 < \epsilon$ for some constant value ϵ for all k. Also, it is possible to use the convergence of some expectation value (i.e. magnetization) calculated between consecutive iterations as a termination criterion. The computation of 1-body and 2-body local expectation values in SU is carried out using the λ weights as local environments. Expectation values illustrations of MPS and PEPS representations in SU are depicted in Fig. 4.4. For a more comprehensive description of SU, I refer the reader to the work of Saeed S. Jahromi and Roman Orus in Ref. [5]. The main point of the SU algorithm is that it is just like the BP equations in the sense that we use the same equations and algorithm that are true for trees on loopy-TN and hope that it is not too bad. It turns out that the SU algorithm gives decent results for a wide range

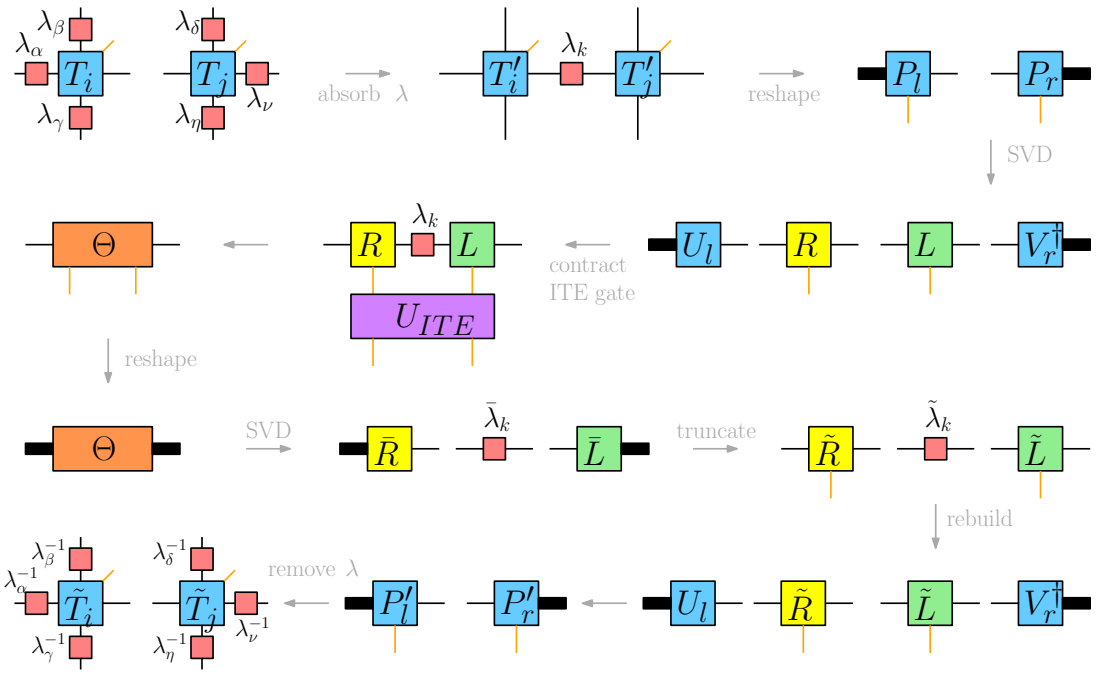
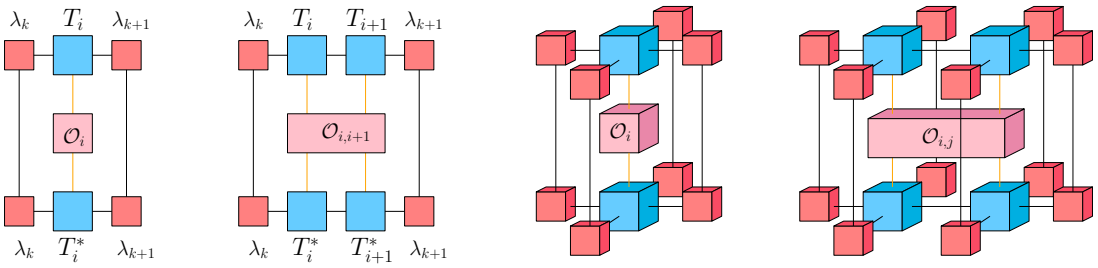


Figure 4.3: Simple Update scheme illustration. For convenience, the illustration is made for the case of PEPS. Also, tensors physical edges are highlighted in orange.

of quantum systems [5], where in chapter 5, we will present some SU numerical results of our own, over the Antiferromagnetic Heisenberg (AFH) model on a 2D lattice.



MPS PEPS

Figure 4.4: Simple Update MPS and PEPS expectation values diagrams from left to right: 1-body MPS, 2-body MPS, 1-body PEPS and 2-body PEPS.

4.3 Belief Propagation Update (BPU)

The Belief Propagation Update (BPU) is an algorithm which we developed for finding ground states of local Hamiltonians using TN. Same as SU, it is also based on the ITE method, with the significant difference of using the BP messages as local environments to truncate the bond dimension, instead of using the SU λ weights. Let us explain how it works. Given some initial TN representation $|\Psi\rangle_{TN}$, we evolve it by applying a local ITE gate over every TN edge. For each ITE iteration over all TN edges, we transform the given TN into a DEFG following 3.3.1. Then, on the resulted DEFG, we apply the BP algorithm to get the double-edge messages. We then use these messages as local environments to truncate every bond dimension following what we call the "BP truncation" scheme, which will be introduced shortly in subsection 4.3.1. The BPU algorithm is fully described in Algorithm 4.3.1.

Algorithm 4.3.1 (Belief Propagation Update (BPU))

1. Given an initial tensor network $|\Psi\rangle$, construct its associated DEFG following 3.3.1.
2. Apply BP on the DEFG following Eqs. (3.32,3.33) until BP convergence criterion is met.
3. Save converged node-to-factor double-edge messages $\{m^*_{n\to f}\}$.
4. Preform an ITE step over every TN edge, resulting in bond dimensions $D' > D$.
5. Use the BP truncation Algorithm 4.3.2 to truncate all the TN bond dimensions.
6. Update all the DEFG factors with the new truncated tensors.
7. Repeat from 2 until TN reaches ground state.

Recall that we showed in section 3.3 that the BP algorithm preformed on DEFG gives exact environments for tree-like graphs and might give reasonable approximations for loopy graphs. Therefore, there is a reason to believe that transforming a given loopy TN into a loopy DEFG and applying BP would result in a set of messages which provide reasonable approximations for the TN local

environments. In the numerical results chapter (5), we present some 2D AFH ground state energies obtained using the BPU algorithm and compare them to the ones obtained using the SU algorithm.

4.3.1 BP Truncation Analysis

After applying the local ITE gate to tensors T_i, T_j connected through a common edge k, we are left with a new TN, $|\psi'\rangle = U_{ij}|\psi\rangle$, with bond dimension $D' > D$. We would like to truncate the bond dimension back to D in the optimal way, while taking into account the environment of the updated tensors $\tilde{T}_i, \tilde{T}_j$. We do that by inserting an operator P on the k edge such that

$$|\psi''\rangle = P|\psi'\rangle \tag{4.8}$$

is our new state which is depicted in Fig. 4.5

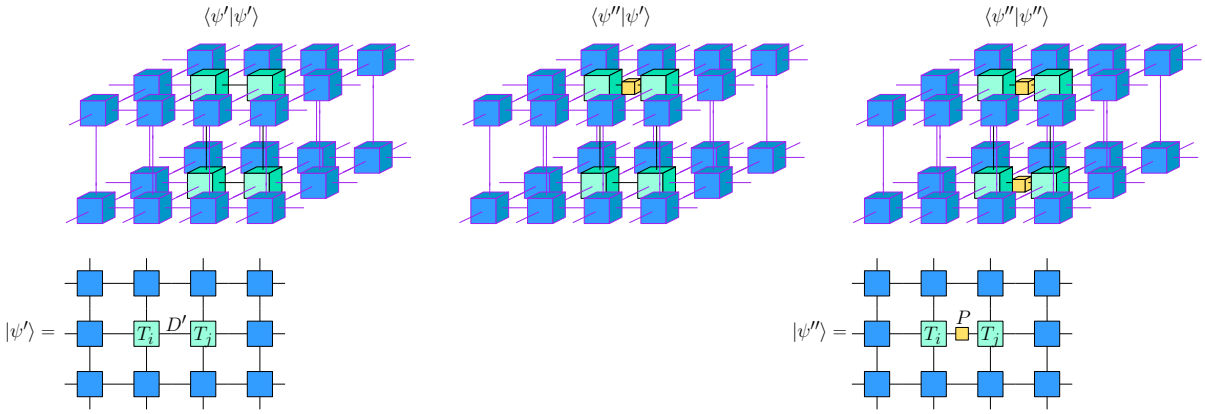


Figure 4.5: Illustration of TN representations $|\psi'\rangle$ and $|\psi''\rangle$.

To get the best approximation, we demand:

1. $\text{rank}(P) \leq D$. This implies that the bond dimension of $|\psi''\rangle$ would be $\leq D$
2. $\||\psi''\rangle\| = 1$
3. $|\langle\psi''|\psi'\rangle|$ is maximal. This will result a maximal overlap between the true state and its approximation.

The BP messages provides us the two tensors A_{ki}, B_{jl} which approximate the effective environment of $\tilde{T}_i, \tilde{T}_j$ (A is the double-edge message from the left and B is the double-edge message from the right), shown in Fig. 4.6.

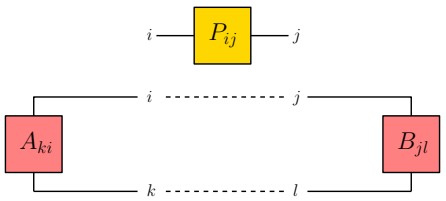


Figure 4.6: Calculating the effective environment of P using the BP messages A, B.

Therefore,

$$\langle \psi' | \psi'' \rangle = \sum_{ijk} A_{ki} P_{ij} B_{jk} = \mathrm{Tr}\left(A^T P B\right) \tag{4.9}$$

$$\langle \psi'' | \psi'' \rangle = \sum_{ijkl} A_{ki} P_{ij} B_{jl} P^*_{kl} = \mathrm{Tr}\left(A^T P B P^\dagger\right) \tag{4.10}$$

where A, B are both Hermitian and non-negative matrices (see 3.3.4). Thus, we can recast our problem as the following maximization problem: Given two non-negative matrices A, B, find a matrix P that maximizes $\mathrm{Tr}\left(A^T P B\right)$ such that

1. $\mathrm{rank}(P) \leq D$
2. $\mathrm{Tr}\left(A^T P B P^\dagger\right) = 1$

Solution: Since A, B are non-negative, we can calculate $A^{-1/2}$, $B^{-1/2}$ and define

$$P \equiv \left(A^T\right)^{-1/2} P_1 B^{-1/2} \tag{4.11}$$

which give us

$$\mathrm{Tr}\left(A^T P B\right) = \mathrm{Tr}\left(\left(A^T\right)^{1/2} P_1 B^{1/2}\right) \tag{4.12}$$

$$= \mathrm{Tr}\left(B^{1/2} \left(A^T\right)^{1/2} P_1\right) \tag{4.13}$$

and

$$\mathrm{Tr}\left(A^T P B P^\dagger\right) = \mathrm{Tr}\left(A^T \left(A^T\right)^{-1/2} P_1 B^{-1/2} B \left(\left(A^T\right)^{-1/2} P_1 B^{-1/2}\right)^\dagger\right) \tag{4.14}$$

$$= \mathrm{Tr}\left(A^T \left(A^T\right)^{-1/2} P_1 B^{-1/2} B \left(B^{-1/2}\right)^\dagger P_1^\dagger \left(\left(A^T\right)^{-1/2}\right)^\dagger\right) \tag{4.15}$$

$$= \mathrm{Tr}\left(P_1 P_1^\dagger\right) \tag{4.16}$$

We define

$$C \equiv B^{1/2} \left(A^T\right)^{1/2} \tag{4.17}$$

Then, the new problem we need to solve is

1. $\max_{P_1} \mathrm{Tr}\left(CP_1\right)$, such that $\mathrm{rank}(P_1) \leq D$
2. $\mathrm{Tr}\left(P_1 P_1^\dagger\right) = 1$

Applying an SVD operation such that $C = USV^\dagger$ where

$$S = \begin{pmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_{D'} \end{pmatrix}, \quad s_1 \geq s_2 \geq ... \geq s_{D'} \geq 0 \tag{4.18}$$

Defining $P_2 \equiv V^\dagger P_1 U$, then $P = \left(A^T\right)^{-1/2} V P_2 U^\dagger B^{-1/2}$, and we get the following problem:

1. $\max_{P_2} \mathrm{Tr}\left(SP_2\right)$, such that $\mathrm{rank}(P_2) \leq D$
2. $\mathrm{Tr}\left(P_2 P_2^\dagger\right) = 1$

The P_2 that maximizes it is diagonal, like S. Therefore, we might write

$$P_2 = \begin{pmatrix} p_1 & & \\ & p_2 & \\ & & \ddots \end{pmatrix} \tag{4.19}$$

where the total number of non-zero p_i elements is at most D, which is the same as the rank of P_2. Then, the maximization problem is:

$$\max_{p_1,...,p_D} \sum_{i=1}^{D} s_i p_i \quad \text{s.t.} \quad \sum_{i=1}^{D} |p_i|^2 = 1 \tag{4.20}$$

and the solution would be

$$p_i = \frac{1}{\mathcal{N}} s_i \qquad \mathcal{N} \equiv \sqrt{\sum_{i=1}^{D} s_i^2} \tag{4.21}$$

To summarize, the BP truncation is as follows:

Algorithm 4.3.2 (BP truncation)

1. Given the A, B message, calculate $C \equiv B^{1/2} \left(A^T\right)^{1/2}$
2. Calculate the SVD of C: $C = USV^\dagger$.
3. Define P_2 to be the diagonal matrix

$$P_2 = \frac{1}{\mathcal{N}} \begin{pmatrix} s_1 & & & & & & \\ & s_2 & & & & & \\ & & \ddots & & & & \\ & & & s_D & & & \\ & & & & 0 & & \\ & & & & & \ddots & \end{pmatrix}, \qquad \mathcal{N} \equiv \sqrt{\sum_{i=1}^{D} s_i^2}$$

4. Calculate P using

$$P = \left(A^T\right)^{-1/2} V P_2 U^\dagger B^{-1/2}$$

Since P_2 is diagonal and contains only D non-zero elements, P would have at most D non-zero singular values. Thus, we can use P to truncate the bond dimension of $\tilde{T}_i, \tilde{T}_j$ from D' back to D.

Chapter 5

Numerical Results

In this chapter, we compare numerical results obtained with the BPU and SU algorithms on a spin-1/2 Antiferromagnetic Heisenberg (AFH) model on a finite PEPS with open boundary conditions. We ran the BPU algorithm to obtain a TN that approximates the AFH ground-state. Then, we ran the SU algorithm to get another TN approximation of the ground-state. We compared the energies of the ground-state we got using BPU to the energies of the ground-state we obtained with SU. To calculate these energies we used the boundary Matrix Product Operator (bMPO) algorithm — which is an algorithm for calculating expectation values using a controlled approximation of the TN environments Ref. [22]. In addition, we compared the energies of the SU/BPU we extracted to the exact energies taken from Ref. [22] obtained using the Full-Update (FU) method. We begin with a simplified description of the bMPO algorithm. Then, we will present the 4×4 and 10×10 PEPS numerical results. The specific details are discussed in the following.

Recall that for general PEPS, the computation of an expectation value or even the norm is known to be a hard problem [21]. Hence only an approximate contraction is possible for already moderate lattice sizes. The bMPO algorithm approximates the two-dimensional TN of an expectation value $\langle\Psi|\mathcal{O}|\Psi\rangle$ or the norm $\langle\Psi|\Psi\rangle$ by means of a succession of one-dimensional MPS - MPO contractions, as sketched in Fig. 5.1.

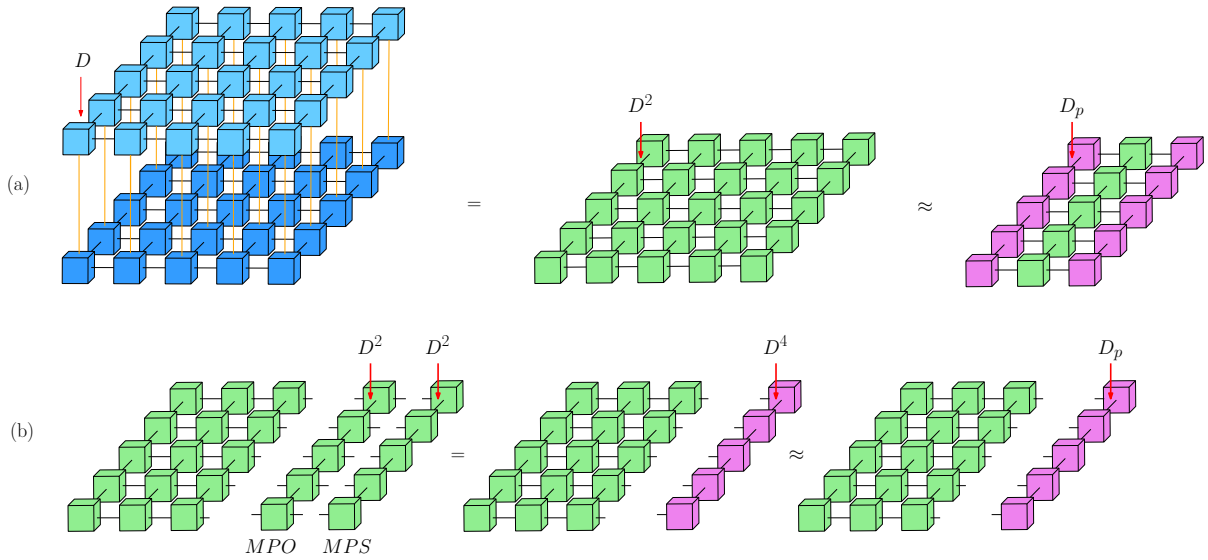


Figure 5.1: Calculating the finite PEPS norm $\langle\Psi|\Psi\rangle$ with bond dimension D by means of the bMPO algorithm. (a) Contracting $\langle\Psi|\Psi\rangle$ physical legs (in orange), Then the intractable computation of "flat" PEPS with bond dimension D^2 is approximated with an efficiently contractible one-dimensional MPS expectation value with bond dimensions D_p. (b) This is done by successively approximating the MPS - MPO contractions of neighboring bulk rows with a new boundary MPS, of bond dimension D_p.

The procedure starts by identifying two opposite sides of the TN with MPS, and each of the intermediate rows with an MPO. Beginning from one of the edges, the contraction of the last row with the neighboring one is then an MPS - MPO product, which can be optimally approximated by a new MPS of fixed bond dimension D_p as illustrated in Fig. 5.1(b). By repeating the procedure from both opposite sides, successive MPS - MPO approximations lead to a representation of both halves of the TN by MPS with bond dimension D_p. Finally, the row in the center is contracted between the two MPS to give the approximate expectation value or norm. At each point of this procedure, the MPS obtained approximates the boundary between the contracted part of the network and the rest. Although in principle, D_p could scale exponentially with the number of rows contracted, in practice, it scales independently of the system's size, which makes this method efficient even for large systems.

The bMPO algorithm is a powerful tool in calculating 2D TN expectations, norms, and tensor environments. In our work, we used it to calculate the energies of the AFH ground-state tensor network approximations obtained with the SU and BPU algorithms. For a more comprehensive description of the bMPO algorithm I refer the reader to Ref. [22].

5.1 Antiferromagnetic Heisenberg (AFH) model

Consider the spin-1/2 AFH model on a square lattice. The AFH Hamiltonian is given by

$$\mathcal{H}_{AFH} = \sum_{\langle i,j \rangle} \boldsymbol{S}_i \cdot \boldsymbol{S}_j \tag{5.1}$$

where $\sum_{\langle i,j \rangle}$ denotes the sum over all nearest-neighbors interactions. The spin operator of site i is given by $\boldsymbol{S}_i = (S_i^x, S_i^y, S_i^z)$, where $S_i^\alpha = \sigma_i^\alpha/2$, and σ_α, $\alpha = x, y, z$ are the three Pauli matrices. We calculated TN approximations of the model's ground-state over a finite 4×4 and 10×10 PEPS for different bond dimensions using the SU and the BPU algorithms. Then, for each one of those TN we calculated the energy per-site by means of the bMPO algorithm and compared the results to the exact energy that was taken from Ref. [22] and calculated with the FU method. Our results compared with the exact ground-state energies are shown in Table 5.1.

AFH PEPS							
		4×4			10×10		
D	D_p	BPU	SU	Exact	BPU	SU	Exact
2	4	−0.54402	−0.54409	−0.54458(2)	−0.61270	−0.61275	−0.61310(2)
	8	−0.54404	−0.54411	−0.54458(2)	−0.61280	−0.61285	−0.61310(2)
3	9	−0.55396	−0.55397	−0.56101(2)	−0.61844	−0.61843	−0.62002(2)
	18	−0.55398	−0.55400	−0.5612(1)	−0.61845	−0.61845	−0.62000(2)
4	16	−0.56273	−0.56279	−0.5738(3)	−0.62369	−0.62371	−0.62636(3)
	32	−0.56278	−0.56284	−0.5739(2)	−0.62378	−0.62380	−0.62637(2)
5	25	−0.56621	−0.56623	−0.57408(1)			
	50	−0.56625	−0.56626	−0.5741(3)			
6	36	−0.56676	−0.56678	−0.57418(2)			
	72	−0.56681	−0.56683	−0.57419(1)			

Table 5.1: Comparison between TN approximated ground-state energies per-site for 4×4 and 10×10 AFH PEPS with open boundary conditions. The ground-state TN approximations of bond dimension D were obtained by means of the SU and BPU algorithms. Then, the energies per-site were calculated using bMPO with $D_p = D^2, 2D^2$ for both SU and BPU ground-state approximations and compared to the exact energies that were taken from Ref. [22].

We ran the bMPO method with $D_p = D^2, 2D^2$ over all simulations, and saw that taking $D_p = 2D^2$ does not deliver significant improvement relative to $D_p = D^2$ in both SU and BPU. Thus, indicates that the bMPO method converged already

for $D_p = D^2$ as empirically found in Ref. [22]. Moreover, the relative errors between SU and BPU for all bond dimensions used, overall simulations, end up to be smaller than 10^{-4}. Therefore, suggesting that the two algorithms converged to similar TN approximations of the AFH ground-state. Although the SU and BPU algorithms obtained similar results, we see in Table 5.1 that both methods still remain far from the exact energy values. This is not surprising considering some previous results obtained with the SU algorithm in [5][6][22] which suggest that this is due to limitations in the update procedure rather than the PEPS ansatz itself. In conclusion, although being far from the exact energy values, the remarkably small relative errors between SU and BPU validate our BPU algorithm as an equal to the SU algorithm in finding ground-state tensor network approximations for local Hamiltonians.

All the numerical results above obtained using the next parameters: The ITE step sizes were taken to be $\tau = [0.5, 0.1, 0.05, 0.01, 0.005]$. The SU and BPU simulations termination criterion was taken with respect to the energy per-site calculated between consecutive time steps and is given by $\|E^{(t+1)} - E^{(t)}\|_1 < \tau \cdot 10^{-5}$. For the DEFG BP algorithm we used two competing termination criteria:

1. maximal number of BP iterations allowed was $t_{max} = 100$.
2. $\|m^{(t+1)} - m^{(t)}\|_1 < 10^{-5}$ for all messages.

whatever came first. We also used messages update dumping of $\gamma = 0.2$ between every two consecutive BP steps, such that

$$\{m^{(t+1)}\} = (1-\gamma)\{m^{(t+1)}\} + \gamma\{m^{(t)}\} \tag{5.2}$$

where $\{m^{(t)}\}$ indicates the set of all BP messages at time step t. The dumping value was chosen with a try and error of many simulations while its aim was to force the convergence of the BP equations into a fixed point. All simulations performed on a Macbook Air laptop with a 1.8 GHz Intel Core i5 processor.

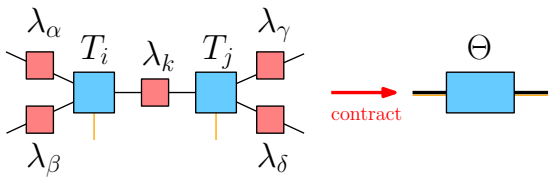
Chapter 6

BP — Trivial SU numerical comparison

In the last chapter, we have seen that both the BPU and SU algorithms give very similar results in calculating ground-states energies of the AFH PEPS on a 2D lattice. Maybe it is not surprising, given that both BP and SU algorithms are exact on tree graphs and deteriorate when loops are introduced. Also, the BP equations are the zeroth-order approximation of calculating graphical models local marginals, where we take the smallest building blocks in the approximation scheme to be single factors. Similarly, the SU algorithm is the most straightforward TN algorithm for evolving TN in the sense that the λ weights are holding individual tensors local environments information, which can also be thought of as some zeroth-order approximation in TN environment calculations. Still, we were looking for some "quantitative measure" that would numerically backup the similarities between the two methods. As both methods are known for environments calculations, we thought it would be interesting to compare them in doing just that, without the imaginary time evolution to get in the way. In this chapter, we present such a comparison. Specifically, we took a random PEPS TN and calculated its 1-body reduced density matrices (RDMs) using the BP algorithm and compared them to the reduced density matrices that we calculated using an algorithm that we call the "trivial SU" algorithm. The trivial SU algorithm, which we describe below, is equivalent to the SU algorithm without the ITE and truncation steps; it only consists of consecutive SVD steps on each edge, and its fixed point corresponds to a canonical representation of the TN we started with. From this canonical representation, the reduced density matrices can be calculated in the same manner done in the regular SU algorithm (see Fig. 4.2).

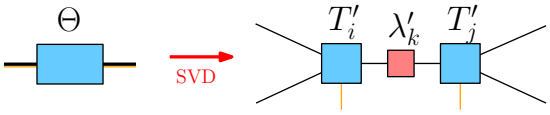
The trivial-SU algorithm Assume we are given a random TN representation $|\Psi\rangle$. Then, the trivial-SU algorithm on $|\Psi\rangle$ is a sequence of SVD maps done on its edges, which resemble the SU steps without the ITE and the truncation. Specifically, given an edge k and its associated tensors T_i, T_j, the SVD map on an edge is described by the following three steps:

1. Contract both tensors with their surrounding environment tensors λ weights into a Θ tensor as follow:

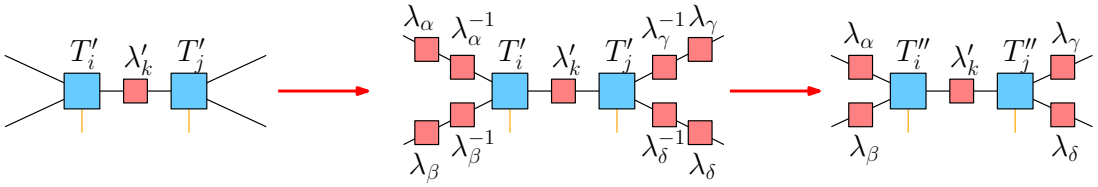


   Note that we fuse all the open legs of the i'th site together to create a super-index, and do the same with all the open legs of the j'th site. The resultant tensor Θ can then be viewed as a *matrix* of two super-indices.

2. Then, preform an SVD operation and de-fuse the super indices to get the new tensors T'_i, T'_j and weights λ'_k



3. Finally, we "pull out" the original environment from T'_i, T'_j by multiplying them with the identity $\lambda \cdot \lambda^{-1} = \mathbb{1}$ over every virtual edge .



It is important to notice that after these 3 steps, the contraction of T''_i, λ'_k, T''_j is exactly equal to the contraction of T_i, λ_k, T_j — so overall, the new TN represents the *same* physical state. In the trivial-SU algorithm we iterate these three steps for all the edges until all tensors and λ weights converge. It is not hard to show that if the TN is a tree, then, these steps converge in a linear time to the canonical form of $|\Psi\rangle$, which corresponds to its Schmidt decomposition along all

edges. In case we are dealing with loopy TN, we just use the same process as if the given TN was a tree. After all tensors and weights converged, we would end up with λ weights that approximate the TN local environments as we mentioned in section 4.2.

To compare trivial-SU with BP, we conducted the next numerical experiment. We drew an initial random 10×10 PEPS, $|\Psi_0\rangle$, with physical bond dimension $d = 2$ and virtual bond dimension $D = 3$ (with λ weights) and used three different algorithms to approximate its 1-body RDMs: trivial-SU, BP and bMPO. We ran the bMPO calculation twice, once with $D_p = 16$ and then with $D_p = 32$ to verify its convergence. Therefore, we ended up with the next four sets of single site RDMs,

$$\{\rho_i^{tSU}\}_{i=1}^n,\ \{\rho_i^{BP}\}_{i=1}^n,\ \{\rho_i^{bMPS(16)}\}_{i=1}^n,\ \{\rho_i^{bMPS(32)}\}_{i=1}^n$$

where ρ_i denotes the i'th particle RDM obtained using the algorithm in the superscript. To measure the distance between two sets A, B of RDMs, we used the next averaged trace distance formula

$$\text{distance}[A, B] \equiv \frac{1}{n}\sum_{i=1}^{n} T(\rho_i^A, \rho_i^B), \tag{6.1}$$

where $T(\rho, \sigma) \overset{\text{def}}{=} \frac{1}{2}\|\rho - \sigma\|_1$ is the trace distance between the two ρ, σ density matrices. We ran the experiment for 100 different random 10×10 PEPS realizations with physical bond dimension $d = 2$ and virtual bond dimension $D = 3$. For each realization we calculated the averaged trace distances between the four sets of RDMs, the results are shown in Fig. 6.1.

Remarkably, we see in Fig. 6.1 that the distance between the trivial-SU and BP is $< 10^{-7}$, suggesting that these two methods converge to the *same* RDMs and the differences we see are only due to machine precision and finite running time. In addition, both methods give reasonable approximations to the "true" RDMs as calculated by the bMPO method. These surprising results led us to search a theoretically fundamental connection between the BP algorithm and the trivial-SU algorithm, that is presented in the next chapter.

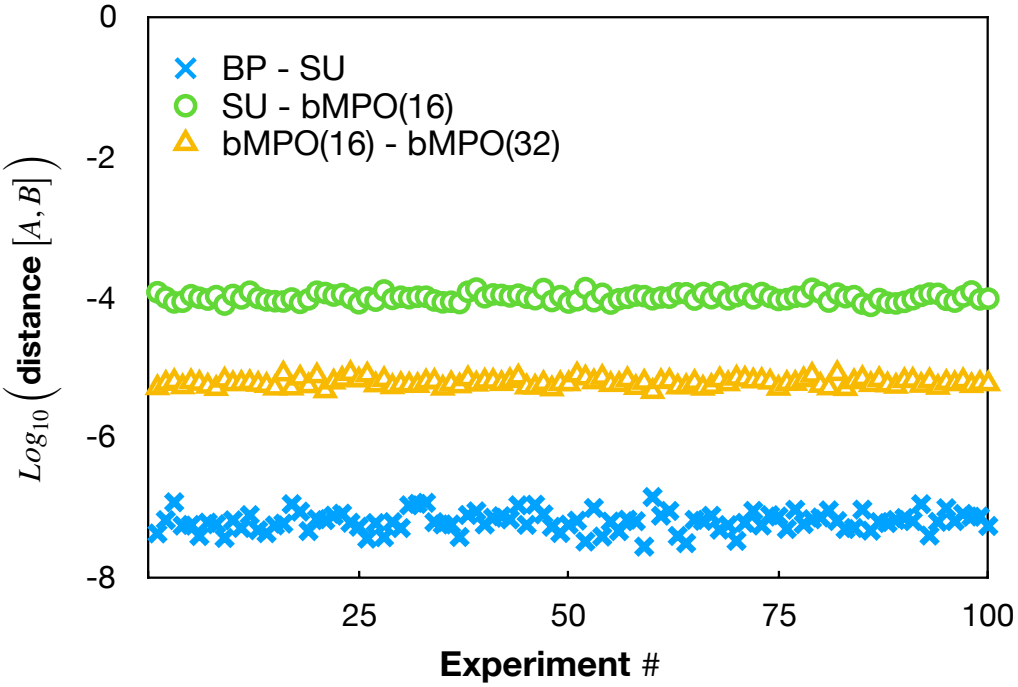


Figure 6.1: The base 10 logarithm of the averaged trace distance between BP, trivial-SU and bMPO RDMs in 100 randomly initialized 10×10 PEPS experiments with physical bond dimension $d = 2$ and virtual bond dimension $D = 3$.

Chapter 7

BP — SU equivalence

In the previous chapters, we have seen a tight connection between the BP algorithm and the so-called trivial-SU algorithm. We showed numerically that given some random PEPS tensor network, both algorithms converge to the same set of single site RDMs. These numerical results led us to believe that there is more here than just pure luck. In this chapter, we present a proof of the equivalence between the BP and trivial-SU algorithms. So, given some PEPS TN, we wrap up our work by proving that its BP fixed point messages and trivial-SU fixed point converge to the same quantum state tensor network representation.

The trivial-SU algorithm and the BP algorithm for TN are two different iterative algorithms for approximate PEPS contraction. They originate from two very different places: the trivial-SU algorithm is a natural algorithm for TN, which relies on the Schmidt decomposition (and SVD), whereas the BP algorithm is a message-passing algorithm for graphical models that originated from inference problems and the Bethe-Peierls approximation. It might, therefore, come as a surprise that these two algorithms are equivalent. In hindsight, this could have been anticipated, as both are exact on trees. Therefore, we prove

Theorem 7.0.1 *Every trivial-SU fixed point corresponds to a BP fixed point such that the local RDMs computed in both methods are identical.*

As a simple corollary, we conclude that if the trivial-SU equations and the BP equations have a unique fixed point, both algorithms will yield the same RDMs. In light of this equivalence, the success of the SU algorithm (with imaginary time evolution) for many models (see, for example, models analyzed in Ref. [5]), is another example of the success of the Bethe-Peierls approximation. Regarding Theorem 7.0.1, we note that in the BP algorithm the TN remains fixed, while

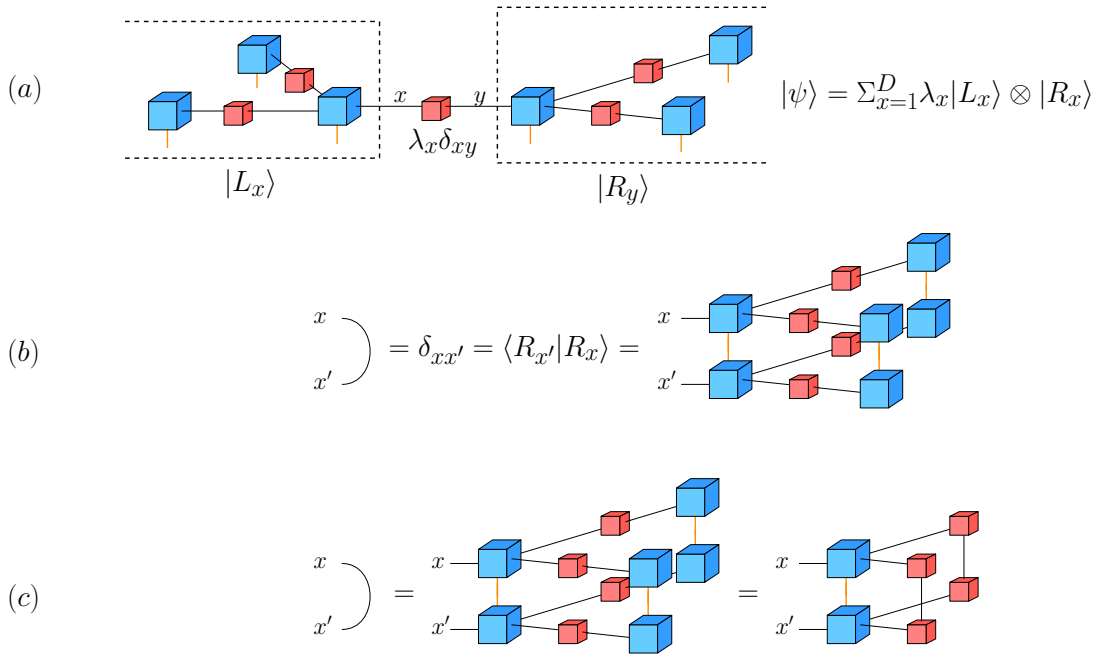


Figure 7.1: The properties of a canonical representation of a tree PEPS. (a) An example of a canonical representation of a tree PEPS and its relation to the Schmidt decomposition. The red squares relate to the diagonal tensors that correspond to the Schmidt λ weights. (b) The orthonormality of the Schmidt bases implies a simple formula for the contraction of the left and right branches. (c) The *quasi-canonical* condition is a local canonical condition on the PEPS tensors. This condition holds up also when the underling PEPS has loops.

the BP messages evolve to a fixed point. In the trivial-SU, there are no messages, but the local tensors that make up the TN evolve until they converge to a quasi-canonical fixed-point, without changing the underlying quantum state. We define the quasi-canonical property for PEPS pictorially in Fig. 7.1. We say that a PEPS TN is in its quasi-canonical representation if it satisfies the local orthonormality condition in Fig. 7.1(c) for every pair of virtual edges x, x'. In both cases, the evolution is done via local steps. Our proof uses two lemma:

Lemma 7.0.2 *Let $\mathcal{T}, \mathcal{T}'$ be two tensor-networks that represent the same state $|\psi\rangle$, such that $\mathcal{T}'$ is obtained from $\mathcal{T}$ using a single trivial-SU step on tensors T_a, T_b and the λ weight between them. Then, every BP fixed-point of $\mathcal{T}$ has a corresponding fixed-point of $\mathcal{T}'$ with the same RDMs, and vice-versa.*

The idea of the proof is to show that the BP fixed point messages of the new TN can be constructed from the BP fixed point messages of the old TN, except for the local place of change, where the messages are adapted to fit the new tensors.

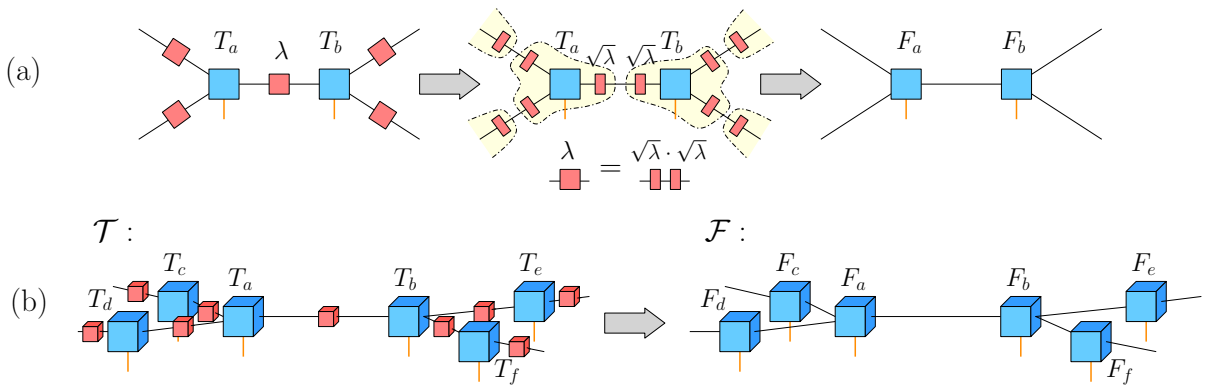
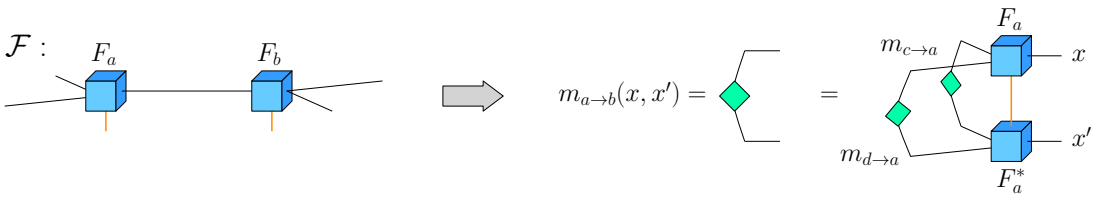


Figure 7.2: (a) Swallowing the λ weights in the T tensors and obtaining an equivalent TN with F tensors. The red squares denote a Simple-Update weight tensor $\lambda_x \delta_{xy}$ and the red rectangulars denote its square root: $\sqrt{\lambda_x}\delta_{xy}$ (b) $\mathcal{T}$ TN to $\mathcal{F}$ TN transformation.

Proof: Assume a trivial-SU step that changes the TN $\mathcal{T}$ to $\mathcal{T}'$ by locally changing the adjacent tensors T_a, λ, T_b to T'_a, λ', T'_b, while keeping the rest of the tensors fixed (see steps in section 6). To simplify the book-keeping, we "swallow" the λ tensors in the T_a, T_b tensors by splitting every λ tensor into $\lambda = \sqrt{\lambda} \cdot \sqrt{\lambda}$ and contracting each $\sqrt{\lambda}$ with its adjacent T_a, T_b tensor, see Fig. 7.2a). We denote the resulting tensor networks by $\mathcal{F}, \mathcal{F}'$, and note that their local tensors are identical except for the F_a, F_b and F'_a, F'_b tensors, which are equal to the T_a, T_b, T'_a, T'_b tensors contracted with the appropriate $\sqrt{\lambda}$ tensors. Also, the BP algorithm performed on $\mathcal{F}, \mathcal{F}'$ would be of messages from factor-to-factor which makes the proof less complicated but with an equal generality.

The $\mathcal{F}, \mathcal{F}'$ BP equation can be obtained by substituting the node-to-factor BP equation into the factor-to-node equation, to end up with a single BP messages equation of factor-to-factor. The fact that the contraction of (T_a, λ, T_b) is equal to the contraction of (T'_a, λ', T'_b) implies that the contraction of (F_a, F_b) is equal to the contraction of (F'_a, F'_b). Now, let $\{m_{a\to b}(x, x')\}$ be fixed-point BP messages of $\mathcal{F}$. We will use these messages to construct the fixed-point BP messages $\{m'_{a\to b}(x, x')\}$ of $\mathcal{F}'$ that give the same RDMs. All messages except for the $a \to b$ and $b \to a$ messages remain the same. The $a \to b$ and $b \to a$ messages are defined by the BP iterative equations using the new tensors F'_a, F'_b so that they will satisfy them. For example, if tensor F_a is connected also to tensors F_c ,F_d in addition to F_b (as illustrated in Fig. 7.2b), then $m'_{a\to b}(x, x')$ is given by the next diagram:



To finish the proof, we need to show that this new set of messages (i) is a BP fixed-point, and (ii) produces the same RDMs according to the BP formula (see Fig. 4.2). Clearly, for adjacent vertices that have nothing to do with a, b, both conditions hold trivially, as the relevant messages and underlying tensors are unchanged. Let us then verify these points for vertices in the vicinity of a, b.

(i): By definition, the $a \to b$ and $b \to a$ messages satisfy the BP equations. So we only need to verify that other messages from a or b (but not between them) satisfy the BP equations. Consider, for example, the message $b \to e$. We need to verify that $m'_{b\to e}(x, x')$ is indeed a BP fixed-point, given as the appropriate expression of $m'_{a\to b}, m'_{f\to b}$ (see Fig. 7.2b for the graph). This is proved in Fig. 7.3 in a series of 5 simple equalities (see the caption for full explanation), which rely on the fact that the original messages are fixed point of the BP equations, and that the contraction of F_a, F_b is equal to the contraction of F'_a, F'_b.

(ii): By definition if we are interested in 2-body RDMs on vertices that are different from both a and b, then the RDM estimate will remain the same because neither the relevant messages, nor the tensors changed. We only need to verify for the RDM ρ_{ab} and RDMs that contain a or b with other adjacent node, such as ρ_{be}. For the former, $\rho_{ab} = \rho'_{ab}$ because it depends on the incoming messages to the a, b nodes (which remain the same), together with the contraction of F'_a, F'_b, which by assumption is identical to that of F_a, F_b. For the latter, the proof uses the same idea as in point (i). Using the assumption that the contraction of F_a, F_b is identical to that of F'_a, F'_b, and that $F'_e = F_e$, it is easy to show that the 3-body RDM ρ_{abe} is identical to that of ρ'_{abe}, from which we deduce that $\rho'_{be} = \rho_{be}$ by tracing out a. This concludes the proof of Lemma 7.0.2.

■

Using this lemma repeatedly along the trivial-SU iterations, we conclude that the BP fixed points of an initial TN is equivalent to those of its quasi-canonical representation. To finish the proof, we show that the BP fixed point of a quasi-canonical PEPS yields the same RDMs as the λ weights do:

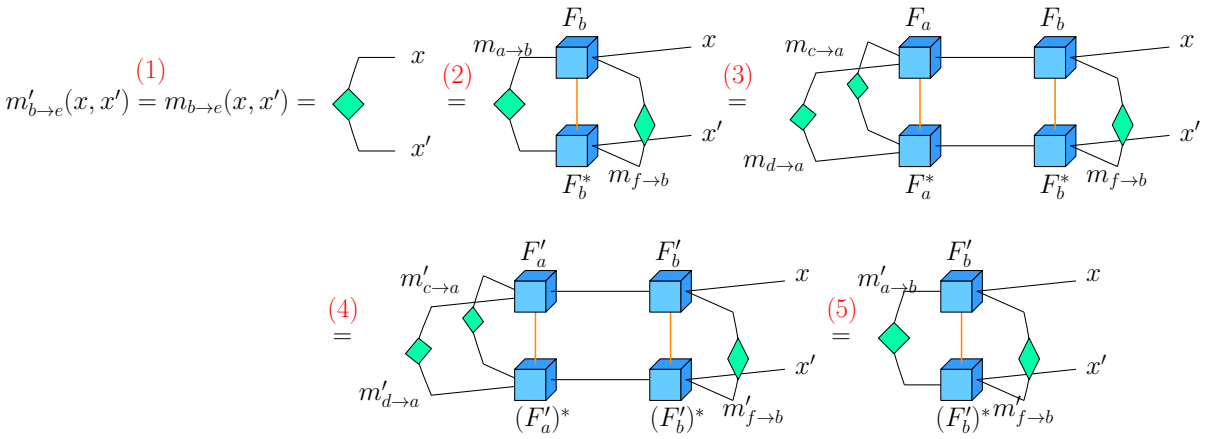


Figure 7.3: Proving that $m'_{b\to e}$ is given by the BP propagation of messages $m'_{a\to b}$ and $m'_{f\to b}$: Equality (1) follows from definition, $m'_{b\to e} = m_{b\to e}$. Then in (2) we use the assumption that $m_{b\to e}$ is a fixed point of the BP equation, and similarly in (3) we use that assumption on $m_{a\to b}$. In (4) we use the fact that the contraction of F_a, F_b is equal to the contraction of F'_a, F'_b, together with the definitions that all the new messages are equal to the old messages, except for the $a \leftrightarrow b$ messages. Finally, in (5) we use the definition of $m'_{a\to b}$ which was designed to satisfy the BP equations.

Lemma 7.0.3 *Given a TN in a quasi-canonical form (i.e., a fixed point of the trivial SU algorithm), it has a BP fixed point that gives the same RDMs estimates as those of the quasi-canonical form based on the λ weights.*

The idea of the proof is that after "swallowing" a $\sqrt{\lambda}$ of each λ tensor in its two adjacent T_a, T_b tensors, we reach a PEPS tensor network for which the messages $m_{a\to b}(x,x') = \lambda_x \delta_{x,x'}$ are a BP fixed-point.

Proof: As in the first lemma, we first define $\mathcal{F}$ to be an equivalent TN in which every λ weight tensor in $\mathcal{T}$ was split into $\sqrt{\lambda}\cdot\sqrt{\lambda}$ and the $\sqrt{\lambda}$ tensors are contracted into the T_a tensors to give the F_a tensors (see Fig. 7.2a). Next we define a set of messages

$$m_{a\to b}(x,x') \stackrel{\text{def}}{=} \lambda_x \delta_{xx'} \tag{7.1}$$

for every two adjacent vertices a, b, where λ is the weight on the ab edge in the original $\mathcal{T}$ TN. We claim that: (i) these messages are BP fixed-point on the $\mathcal{F}$ TN, and (ii) they give the same 2-body RDMs as those of the trivial-SU method of quasi-canonical $\mathcal{T}$. Both claims are immediate. Claim (i) follows by writing the BP equation for the $a \to b$ message in terms of the F_a tensor, and noticing that this expression is equal to the $\lambda_x \delta_{xx'}$ using canonical condition on $\mathcal{T}$. This

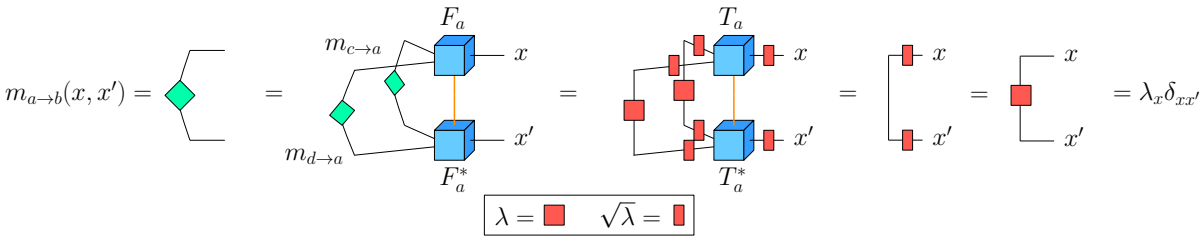


Figure 7.4: Illustration of the proof for claim (i). Following these equalities, we see that Eq. (7.1) result in a BP fixed-point.

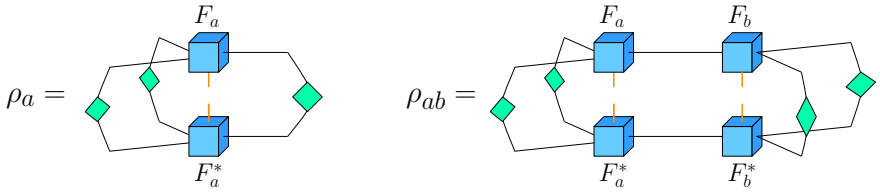


Figure 7.5: Calculating 1 and 2-body RDMs using BP messages.

is illustrated in Fig. 7.4. Claim (ii) follows from definitions of the 2-body RDMs of the BP method (see Fig. 7.5) and the SU method (see Fig. 4.2). ■

From these two Lemmas, the proof of Theorem 7.0.1 follows in a straight forward manner. This chapter concludes the equivalence of the trivial-SU algorithm and the BP equations which paves the way for the usage of more complicated PGM algorithms as contraction algorithms in the TN framework.

Chapter 8

Discussion

This dissertation presents our work as a series of steps, where we started with the idea of constructing a new kind of TN algorithm (BPU) based on the BP equations that would be superior to the well known SU algorithm. When we found that our new algorithm gives the same results as the SU algorithm in calculating TN local expectations, we tested numerically how similar these two algorithms actually are. The numerical results we got led us to suspect that the similarity between these two algorithms is much deeper than simple coincidence in the sense that they are mathematically equivalent. Finally, we proved that equivalence, which opens the door for importing more complicated PGM algorithms to the TN world.

A possible direction along these lines is to improve the accuracy of the BPU algorithm by following the work of Yedidia et al. in Ref. [1]. In this work, the authors significantly improve the approximations of local marginals (beliefs) by introducing the framework of Region Graphs (RG) and the Generalized Belief Propagation (GBP) algorithms. This method's general idea is to cluster neighboring factors and run a variant of the BP algorithm on these "coarse-grained" degrees of freedom. Intuitively, this clustering performs the summation over the smallest loops exactly, thereby reducing the primary source of errors in the BP method. Although this method produces more accurate beliefs, it comes with a price; the GBP algorithm's computational resources are significantly higher than those of the BP algorithm, although this price is not always that high due to the fact that GBP often converges in much fewer iterations than BP.

As an example of the significant improvement in accuracy of the GBP algorithm over the BP algorithm, consider a numerical benchmark from the original

paper of Yedidia et al [1]. In that work, the authors compared the BP and GBP methods by evaluating local magnetizations of a 10×10 spin-glass model. They showed that the GBP algorithm performs significantly better than the BP algorithm on this kind of model, see Fig. 8.1. Therefore, there is a good reason to believe that running the GBP algorithm on our DEFG would drastically improve the accuracy of TN environments for the BPU algorithm.

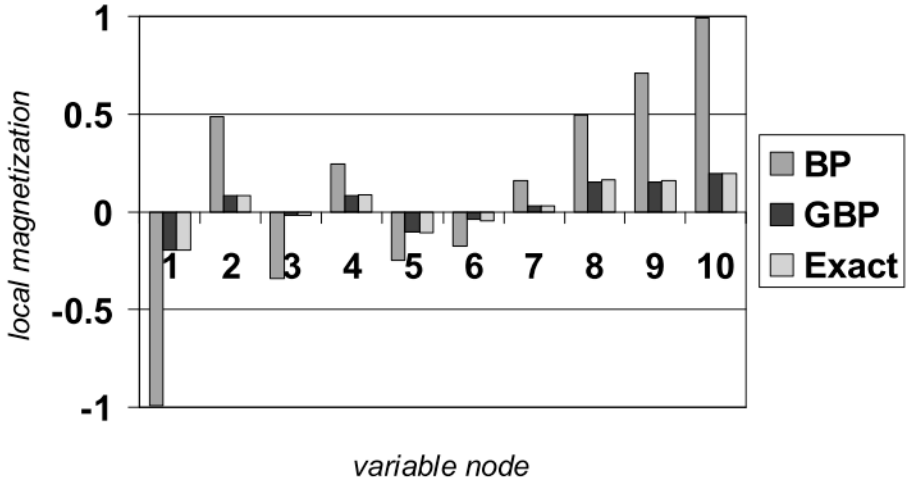


Figure 8.1: The local magnetization for 10 variable nodes in a 10×10 spin-glass with random magnetic fields, as computed exactly, and using ordinary BP or GBP. This figure was taken from the work of Yedida et al. in Ref. [1].

A second possible theoretical direction would be to understand better the complex Bethe free-energy's physical and mathematical role derived from the BP equations in the DEFG framework where we have complex-valued factors. In particular, we know that for tree-TN, it is related to the Schmidt decomposition. Can we somehow relate it to the underlying entanglement structure also when the underlying graph has loops? Another interesting question is whether the BP equations can be used to analytically analyze models for which the ground state is a known PEPS, such as the AKLT model. Finally, we note that unlike the trivial-SU algorithm, our BP scheme easily generalizes to mixed states described by PEPO tensor networks, for which there is no natural Schmidt decomposition.

Bibliography

[1] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005. doi:10.1109/TIT.2005.850085. (document), 1, 1, 3, 3.2, 3.2, 8, 8, 8.1

[2] Guifré Vidal. Efficient simulation of one-dimensional quantum many-body systems. *Physical Review Letters*, 93(4):040502–1, 2004. arXiv:0310089v1, doi:10.1103/PhysRevLett.93.040502. 1, 2.1

[3] J. Jordan, R. Orús, G. Vidal, F. Verstraete, and J. I. Cirac. Classical simulation of infinite-size quantum lattice systems in two spatial dimensions. *Physical Review Letters*, 101(25):2–5, 2008. arXiv:0703788v4, doi:10.1103/PhysRevLett.101.250602. 1, 2.1

[4] Philippe Corboz, Román Orús, Bela Bauer, and Guifré Vidal. Simulation of strongly correlated fermions in two spatial dimensions with fermionic projected entangled-pair states. *Physical Review B - Condensed Matter and Materials Physics*, 81(16):1–25, 2010. arXiv:arXiv:0912.0646v2, doi:10.1103/PhysRevB.81.165104. 1, 2.1

[5] Saeed S. Jahromi and Román Orús. Universal tensor-network algorithm for any infinite lattice. *Physical Review B*, 99(19), 2019. arXiv:arXiv:1808.00680v3, doi:10.1103/PhysRevB.99.195105. 1, 2.1, 4, 4.2, 5.1, 7

[6] Yi Zheng and Shuo Yang. Loop update for infinite projected entangled-pair states in two spatial dimensions. pages 1–5, 2019. URL: http://arxiv.org/abs/1906.04085, arXiv:1906.04085. 1, 4, 5.1

[7] G. Evenbly and G. Vidal. Tensor Network Renormalization. *Physical Review Letters*, 115(18):1–6, 2015. doi:10.1103/PhysRevLett.115.180405. 1, 2.1

[8] F. Verstraete, V. Murg, and J. I. Cirac. Matrix product states, projected entangled pair states, and variational renormalization group methods for

quantum spin systems. *Advances in Physics*, 57(2):143–224, 2008. doi:10.1080/14789940801912366. 1, 2.1, 2.4.1

[9] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical Review Letters*, 91(14):1–4, 2003. arXiv:0301063v2, doi:10.1103/PhysRevLett.91.147902. 1, 2.1

[10] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac. Matrix product state representations. *Quantum Information and Computation*, 7(5-6):401–430, 2007. arXiv:0608197v2. 1, 2.1

[11] E. Miles Stoudenmire and David J. Schwab. Supervised Learning with Quantum-Inspired Tensor Networks. pages 1–12, 2016. URL: http://arxiv.org/abs/1605.05775, arXiv:1605.05775. 1, 3

[12] Mario Collura, Luca Dell'Anna, Timo Felser, and Simone Montangero. On the descriptive power of Neural-Networks as constrained Tensor Networks with exponentially large bond dimension. (ii):1–9, 2019. URL: http://arxiv.org/abs/1905.11351, arXiv:1905.11351. 1, 3

[13] Jing Chen, Song Cheng, Haidong Xie, Lei Wang, and Tao Xiang. Equivalence of restricted boltzmann machines and tensor network states. *Physical Review B*, 97(8):085104, 2018. 1, 3

[14] Marc Mezard, Marc Mezard, and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009. 1, 3.2

[15] Michael Irwin Jordan. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998. 1, 3

[16] Brendan J Frey, J Frey Brendan, and Brendan J Frey. *Graphical models for machine learning and digital communication*. MIT press, 1998. 1

[17] Elina Robeva and Anna Seigal. Duality of graphical models and tensor networks. *Information and Inference: A Journal of the IMA*, 8(2):273–288, 2019. arXiv:arXiv:1710.01437v1, doi:10.1093/imaiai/iay009. 1

[18] Michael X. Cao and Pascal O. Vontobel. Double-Edge Factor Graphs: Definition, Properties, and Examples. 2, 2017. URL: http://arxiv.org/abs/1706.00752, arXiv:1706.00752. 1, 3, 3.3

[19] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014. arXiv:arXiv:1306.2164v3, doi:10.1016/j.aop.2014.06.013. 2.1, 2.4.1

[20] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002. 2.2

[21] Norbert Schuch, Michael M. Wolf, Frank Verstraete, and J. Ignacio Cirac. Computational Complexity of Projected Entangled Pair States. *Physical Review Letters*, 98(14):140506, apr 2007. URL: https://link.aps.org/doi/10.1103/PhysRevLett.98.140506, doi:10.1103/PhysRevLett.98.140506. 2.4.1, 5

[22] Michael Lubasch, J. Ignacio Cirac, and Mari Carmen Bañuls. Unifying projected entangled pair state contractions. *New Journal of Physics*, 16(1), 2014. doi:10.1088/1367-2630/16/3/033014. 2.4.1, 4, 5, 5, 5.1, 5.1

[23] G Cooper. Computational Complexity of probabilistic inference using Bayesian belief networks (research note). *Machine Learning*, 42(1990):393–405, 1990. 3.1

[24] Masuo Suzuki. General theory of fractal path integrals with applications to many-body theories and statistical physics. *Journal of Mathematical Physics*, 32(2):400–407, 1991. 4.1

[25] Hong-Chen Jiang, Zheng-Yu Weng, and Tao Xiang. Accurate determination of tensor network state of quantum lattice models in two dimensions. *Physical review letters*, 101(9):090603, 2008. 4.2

נומריות של חישובי אנרגיה עבור מצבי יסוד של מודל הייזנברג אנטיפרומגנטי על שריג דו מיימדי ריבועי אשר התקבלו בעזרת Belief Propagation Update ואשווה אותן לתוצאות אשר התקבלו על יד אלגוריתם ידוע ובעל שימוש רב בעולם רשתות הטנזורים אשר נקרא Simple Update (SU). לאחר מכן, בפרק 6 אתן הסבר בצורת ניסוי נומרי לדמיון הרב בין התוצאות אשר התקבלו בעזרת שני האלגוריתמים המוזכרים לעיל אשר מתבסס על עיקרון הדואליות העומד בין משוואות Belief Propagation לבין השימוש באלגוריתם Simple Update טריוויאלי. לבסוף, בפרק 7, אתן הוכחה מתמטית לשקילות בין אלגוריתם ה-BP לבין אלגוריתם ה-SU הטריוויאלי שתחתום את העבודה שלנו על הקשר בין רשתות טנזורים למודים גרפים הסתברותיים. בפרק 8, אסכם ואציג מספר שאלות מעניינות להמשך ובתוכן את השימוש בעבודתו של ידידיה [2] על Generalized Belief (GBP) Propagation על מנת לנסות ולשפר את האלגוריתם Belief Propagation Update.

תקציר

רשתות טנזורים (Tensor Networks) ומודלים גרפים הסתברותיים (Probabilistic Graphical Models) הן שיטות השייכות לשני תחומים נפרדים, כאשר שתיהן עושות שימוש בהצגה גרפית של מערכות מרובות דרגות חופש על מנת להבין את המבנה הלוקאלי של המערכות והתלות בין חלקיהן השונים של מערכות אלו. בעשור האחרון, נעשה שימוש נרחב ברשתות טנזורים בעיקר בתחומי מחקר פיזיקאליים כגון: פיזיקה של חומר מעובה, קוסמולוגיה, אינפורמציה קוונטית ועוד. לעומת זאת, במודלים גרפים הסתברותיים נעשה שימוש בעיקר בתחומים מעולם מדעי המחשב כגון: ראיה ממוחשבת, עיבוד אותות, למידת מכונה, עיבוד שפה טיבעית ועוד. כמו כן, מודלים גרפים הסתברותיים משמשים גם ככלי למידול מערכות מרובות חלקיקים בעיקר בפיזיקה סטטיסטית. בשנים האחרונות מחקר רב מוקדש למציאת הקשר בין שני העולמות של מודלים גרפים הסתברותיים ורשתות טנזורים על מנת לנסות ולייבא אלגוריתמים בין השניים כדי לנצל את יתרונותיו של האחד באחר.

שאלה רבת משקל בעולם רשתות הטנזורים היא, כיצד ניתן לבצע כיווץ (contraction) יעיל של רשתות טנזורים במיימדים גבוהים (גדולים ממימד אחד). במימדים גבוהים, כיווץ מדויק של רשתות טנזורים דורש כמות אקספוננציאלית של כוח חישוב ולכן בלתי אפשרי עבור מערכות גדולות. בעשור האחרון, נעשה מחקר נרחב לשם מציאת פתרון לבעיה זו אשר הוביל לפיתוח מספר רב של שיטות ואלגוריתמים כגון: Full Update, Fast Full Update, Loop Update, Simple Update ועוד רבים. כל האלגוריתמים הללו נמצאים על הרצף שבין דיוק ההצגה הקוונטית לבין יעילות החישוב. בעולם הקלאסי של מודלים גרפים הסתברותיים אנו עומדים בפני בעיה דומה. בהינתן פוקציית התפלגות כלשהי, על מנת לחשב את פונקציית ההסתברות השולית של משתנה מקרי מסוים, נידרש לסכום כמות אקספוננציאלית של פרמטרים. בעולם מודלים הגרפים ההסתברותיים, לבעיה זו קיים פתרון אשר עושה שימוש באלגוריתמי שליחת הודעות איטרטיביים ובינהם אלגוריתם ספציפי אשר נקרא Belief (BP) Propagation. אלגוריתם BP עושה שימוש בשליחת הודעות על גרפים על מנת לחשב פונקציות הסתברות שוליות של פונקציית התפלגות כלשהי באופן יעיל.

בעבודת מחקר זו, אתאר את השימוש שעשינו במודל גרפי הסתברותי חדש בשם Double (DEFG) Edge Factor Graphs אשר תומך בהתנהגות הקוונטית של רשתות טנזורים. בנוסף, אראה כיצד תחת המסגרת של DEFG ניתן לרתום את יעילותו של BP על מנת לחשב כיווצים של רשתות טנזורים במימדים גבוהים באופן יעיל. לאחר מכן, אציג אלגוריתם חדש אשר נקרא Belief (BPU) Propagation Update ומשמש למציאת קירובים של רשתות טנזורים עבור מצבי יסוד של מערכות מרובות חלקיקים המתוארות על ידי המילטוניאנים בעלי אנטרקציות "שכנים קרובים". לבסוף, אראה שבעצם אלגוריתם ה-BP בעצם שקול לאחד מאלגוריתמי רשתות הטנזורים הנפוץ ביותר אשר נקרא Simple Update ואתן הוכחה מתמטית לשקילות זו.

מבנה עבודת המחקר הוא כדלהלן: בפרק 2 אתן סקירה כללית על רשתות טנזורים ואציג באופן מפורט את בעיית הכיווץ ברשתות טנזורים במימדים גבוהים. בפרק 3 אתאר באופן מקוצר את המודל הגרפי ההסתברותי Factor Graphs (FG) אשר היווה בסיס לפיתוח השיטה בעזרת המודל הגרפי ההסתברותי Double Edge Factor Graphs ששימש אותנו בפיתרון בעיית הכיווץ ברשתות טנזורים. לאחר מכן, בפרק 4 נציג כיצד רתמנו את האלגוריתם Belief Propagation על המודל הגרפי Double Edge Factor Graphs לטובת פיתוח האלגוריתם Belief Propagation Update. בפרק 5 אציג תוצאות

המחקר בוצע בהנחייתו של פרופסור איתי ארד, בפקולטה לפיסיקה.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

שימוש באלגוריתם שליחת הודעות איטרטיבי על מנת למצוא קירובים של רשתות טנזורים למצבי יסוד של מערכות מרובות חלקיקים

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בפיסיקה

רוי אלקבץ

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
אדר התש"ף חיפה מרץ 2020

שימוש באלגוריתם שליחת הודעות איטרטיבי על מנת למצוא קירובים של רשתות טנזורים למצבי יסוד של מערכות מרובות חלקיקים

רוי אלקבץ