Belief propagation for general graphical models with loops

Pedro Hack,[1, 2, *] Christian B. Mendl,[1, †] and Alexandru Paler[3, ‡]

[1] *Technical University of Munich, Germany*
[2] *German Aerospace Center, Germany*
[3] *Aalto University, Finland*

Belief Propagation (BP) decoders for quantum error correcting codes are not always precise. There is a growing interest in the application of tensor networks to quantum error correction in general and, in particular, in degenerate quantum maximum likelihood decoding and the tensor network decoder. We develop a unified view to make the generalized BP proposal by Kirkley et. al explicit on arbitrary graphical models. We derive BP schemes and provide inference equations for BP on loopy tensor networks and, more generally, loopy graphical models. In doing so we introduce a tree-equivalent approach which allows us to relate the tensor network BlockBP to a generalized BP for loopy networks. Moreover, we show that the tensor network message passing approach relies essentially on the same approximation as the method by Kirkley. This allows us to make tensor network message passing available for degenerate quantum maximum likelihood decoding. Our method and results are key to obtaining guidelines regarding how the exchange between complexity and decoding accuracy works between BP and tensor network decoders. Finally, we discuss how the tree-equivalent method and the method by Kirkley can justify why message scheduling improves the performance of BP.

I. INTRODUCTION

Graphical models are used as error correction tools in classical and quantum error correction [1–6]. In both of them, a graphical model contains all the available information that one has in order to correct errors. This information, however, is only implicitly available and has to be brought forward via some computation. It is in this computation where the complexity issue, otherwise known as the decoding problem, lies.

The most extended approach to the decoding problem is belief propagation(BP) [1], an algorithm introduced in cognitive sciences [7] that allows to compute quantities that are relevant to decoding, like the partition function, internal energy, Shannon entropy or marginal over a few variables, in a computationally non-expensive way.

Provided the graphical model under consideration is a tree, BP achieves exact results [1, 7]. Despite the fact that BP has been empirically shown to work somewhat well for graphical models that locally resemble trees [8], the issue remains that the exactness deteriorates as the graphical models becomes more loopy.

To obtain a method that improves on BP whenever it is not accurate, a natural approach is to try to better account correlations by computing them exactly, that is, to use more computational power in order to obtain accuracy. The first such attempt was named generalized BP [9], an approach that was celebrated when it was introduced [10–12], and that is still pursued, even within the quantum error correction community [13].

The main problem with generalized BP is that it does not provide a specific recipe, but rather a general approach that ought to be tuned to the concrete case in consideration. This was realized by Kirkley, Cantwell and Newmann [14, 15], which lead them to the development of a new method (in the following, the **KCN-approach** or **KCN-method**) that, although following an idea similar to generalized BP, provides an explicit construction. Despite its success, however, the method has the drawback that it only applies to networks.

A. Motivation

Recently, and given the growing interest in the application of tensor networks to quantum error correction in general [5, 16–20] and in **degenerate quantum maximum likelihood decoding** [3, 6] and the **tensor network decoder** [21–25] in particular, approaches similar to KCN-method have been introduced in the context of tensor networks.

KCN-approaches follow the basic idea already in generalized BP in two different ways: either they construct a new tensor network by grouping tensors together, like BlockBP does [26, 27], or, in the spirit of the KCN-method, they construct several directed graphs by grouping tensors together and combine the results given by them when doing inference, like **tensor network message passing** does [28]. Here, we relate these two sorts of approaches and extend them to arbitrary graphical models.

B. Contribution

First, we introduce the **tree-equivalent method** (and its inference equations), an approach for networks that is closely related to BlockBP but follows the graph decomposition in the KCN-method. We relate the tree-

* pedro.hack@tum.de
† christian.mendl@tum.de
‡ alexandru.paler@aalto.fi

equivalent method in terms of accuracy and complexity to the KCN-method (Section II B).

Second, we extend both the tree-equivalent and the KCN-method to tensor networks (Section IV) and then to graphical models (Section V). We provide inference equations for both of them. As a result, we improve on tensor network message passing by providing an algorithm that works for any tensor network and obtaining inference equations for global quantities. This results in the method being useful for degenerate quantum maximum likelihood decoding [3, 27] (Section VI A).

We also improve on the relation between graphical models and its two simplest subfamilies (Appendix D), provide a less complex generalization of the KCN-method to arbitrary graphical models given some symmetry properties (Appendix C) and use the KCN-method to justify the improvement on BP given by scheduling in some instances (Appendix VI C).

II. BACKGROUND

A. Graphical models

Take a bipartite graph $\mathcal{G} = (\mathcal{F} \cup \mathcal{V}, \mathcal{E})$ whose edges $\mathcal{E}$ join elements in $\mathcal{F}$ with elements in $\mathcal{V}$. A graphical model on $\mathcal{G}$ is a function

$$P(x_1, \dots, x_n) = \frac{1}{Z} \prod_{a \in \mathcal{F}} f_a(x_{\partial a}), \tag{1}$$

that is defined via a set of functions $\{f_a\}_{a\in\mathcal{F}}$, $f_a : \{x_i\}_{i\in\partial a} \to \mathbb{C}$ (also known as **factors** or potentials), a set of variables $\{x_i\}_{i\in\mathcal{V}}$ and a normalization constant or partition function

$$Z = \sum_{x_1,\dots,x_n} \prod_{a\in\mathcal{F}} f_a(x_{\partial a}). \tag{2}$$

In particular, for all $a \in \mathcal{F}$, the factor f_a depends on x_i provided there exists some $e \in \mathcal{E}$ joining them and we use the notation ∂a and $x_{\partial a}$ for the set of variables that f_a depends on and for a specific configuration of them, respectively. In fact, we will indistinctly refer to f_a as a for all $a \in \mathcal{F}$. Similarly, when dealing with a variable x_i, we will indistinctly refer to it by its associated node $i \in V$, and use the notation ∂i for the set of functions that depend on x_i.

We will use the following definitions and conventions:

- Given a graphical model P, we call its associated graph $\mathcal{G}$ the **factor graph** or the **Tanner graph** [1];
- all the considered graphical models are connected, such that the associated factor graph is connected; otherwise, we simply apply our analysis to each of its connected components;
- we assume all graphs to be undirected, unless explicitly stated otherwise;
- each variable x_i takes values from the same finite set X fulfilling $|X| \geq 2$ for simplicity;
- in the context of error correction, we study the restricted case where $f_a : \{x_i\}_{i\in\partial a} \to \mathbb{R}_{\geq 0}$ for all $a \in \mathcal{F}$, the so-called **probabilistic graphical models** [1, 7, 29]. Nonetheless, we consider a specific case of interest [19, 23, 25, 30] where $f_a : \{x_i\}_{i\in\partial a} \to \mathbb{C}$ in Appendix G.

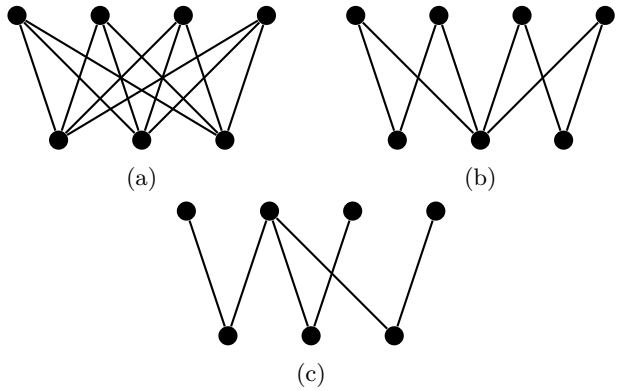


FIG. 1: The classes of graphical models we consider here: (a) General graphical model, (b) network, and (c) tensor network.

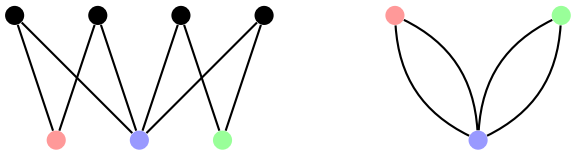
We consider the following classes of graphical models (Figure 1):

- networks [14] (also known as graphical models with pairwise potentials [9]) are graphical models whose functions $\{f_a\}_a$ depend at most on two variables, $|\partial a| \leq 2$ for all $a \in \mathcal{F}$;
- tensor networks [31, 32] are graphical models such that, for each variable x_i, there are at most two functions that depend on x_i, $|\partial i| \leq 2$ for all $i \in \mathcal{V}$

B. The KCN-Method: BP for networks with loops

Since the functions that compose a network P only depend on at most two variables, P admits a simple representation via a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where we associate one node in $\mathcal{V}$ to each variable in P and one edge in $\mathcal{E}$ to each function in P. Moreover, a node is an endpoint of an edge whenever the function associated to the edge depends on the variable associated to the node. We call $\mathcal{G}$ the simplified factor graph associated to P.

An example of a factor graph and its associated simplified factor graph can be found in Figure 2. Note that, in the following, we assume we are given a network whose functions depend exactly on two variables. If there were



(a) (b)

FIG. 2: The factor graph of a network (a), and its associated simplified factor graph (b).

some functions depending on a single variables, these can be naturally incorporated into the scheme [14].

In order to introduce their BP equations on a network, Kirkley et al. [14] define the following neighborhoods (i.e. subgraphs of the simplified factor graph $\mathcal{G}$) given some parameter $\ell_0 \in \mathbb{N}$, which we call the KCN-parameter, and some variables $i, j \in \mathcal{V}$:

(i) The neighborhood around variable i, $N_i^{\ell_0}$, includes node i plus the edges that have it as endpoint and the other endpoints of its edges (i.e. the nearest neighbors of i in $\mathcal{V}$), together with the edges and nodes that belong to a path of length $\ell_0 - 2$ or less connecting two nearest neighbors of i. (Note that we use the parameter ℓ_0 in $N_i^{\ell_0}$ with a slightly different different meaning compared the original notation in [14]. In particular, what we call $N_i^{\ell_0}$ corresponds to $N_i^{\ell_0-2}$ in the notation they use.)

(ii) The neighborhood difference from variable i to j, $N_{i\setminus j}^{\ell_0}$, consists of i together with all the edges that belong to $N_i^{\ell_0}$ and are not included in $N_j^{\ell_0}$ plus the nodes at their ends.

In what follows, for simplicity, we may drop the parameter ℓ_0 and simply write N_i or $N_{i\setminus j}$. Note that we, once we choose the parameter ℓ_0, we use it to construct all the neighbourhoods.

Using these neighborhoods, Kirkley et al. [14] proposed a variation of BP, the KCN-method or KCN-approach, that uses the set of messages

$$\begin{aligned} &\{m_{i\to j}^{(t)}\}_{i\in V, j\in N_i, t\geq 0}, \text{ where} \\ &m_{i\to j}^{(t)} : X \to \mathbb{R}_{\geq 0}. \end{aligned} \tag{3}$$

In their variation, the messages are uniformly initialized,

$$m_{i\to j}^{(0)}(x_i) \equiv 1/|X| \text{ for all } x_i \in X, \tag{4}$$

and are updated, for $t \geq 0$, according to the following equation:

$$m_{i\to j}^{(t+1)}(x_i) \equiv \frac{1}{M_{i\to j}^{(t+1)}} \sum_{x_{N_{i\setminus j}}\setminus x_i} \prod_{a\in N_{i\setminus j}} f_a(x_{\partial a}) \prod_{k\in N_{i\setminus j}} m_{k\to i}^{(t)}(x_k) \text{ for all } x_i \in X, \tag{5}$$

where $M_{i\to j}^{(t+1)}$ is a normalization constant

$$M_{i\to j}^{(t+1)} \equiv \sum_{x_{N_{i\setminus j}}} \prod_{a\in N_{i\setminus j}} f_a(x_{\partial a}) \prod_{k\in N_{i\setminus j}} m_{k\to i}^{(t)}(x_k), \tag{6}$$

and $x_{N_{i\setminus j}}$ denotes the set of variables in $N_{i\setminus j}$. (Note that, in general, the normalization of messages in BP schemes is not an actual theoretical necessity, but a practical addition to avoid numerical underflow [7, 33]. Preventing this will justify the appearance of normalization constants in all the update equations that follow.) Note also that, in the following, we will give preference to the compact tensor notation $\mathrm{tr}(\cdot)$ [31] to avoid explicitly writing the variables in the sum like those in Eq. (5). Similarly, we denote by $\mathrm{tr}_{\setminus A}(\cdot)$ the trace over the variables outside of a set A.

Once the update equations have converged (or we have ran out of iterations), Kirkley et al. [14] provide inference equations through which we can compute quantities of interest like marginals, the internal energy, Shannon entropy or the partition function.

Importantly, whenever the length of the loops in $\mathcal{G}$ is bounded by the KCN-parameter ℓ_0, the inference equations provide exact values for the computed quantities. This is the case since, whenever the bound holds, we can use the neighborhoods to construct a tree equivalent to the original network.

Note that, in case the KCN-parameter is smaller than the largest loops length, the update equations are still well defined, although they are no longer exact. Still, these equations have been reported to give good results in this context [14].

III. A TREE-EQUIVALENT APPROACH TO NETWORKS

We can think of the KCN-method as computing over several trees (one per node in $\mathcal{G}$) and then, when doing global inference, averaging over the inference results each tree provides. To explain this into more detail, we show how to construct a tree starting from each individual node (Section III A) and discuss how this method is related to the KCN-approach (Section III B).

A. Construction method

In order construct a tree with parameter ℓ_0, we can use the following procedure:

(Pi) Pick randomly a node $i_0 \in \mathcal{V}$, which in the following we call the **seed**, and take $A = \{i_0\}$.

(Pii) While there exist vertices in $N_{i_0} \setminus A$ that are connected to vertices outside of N_{i_0}, pick randomly any such that a vertex $i \in N_{i_0} \setminus A$ and add it to A.

(Piii) Take as i the first element in A such that there exist vertices in $N_{i\setminus p(i)} \setminus A$ that are connected to vertices outside of $N_{i\setminus p(i)}$, where $p(i)$ is the **parent** of i in A (that is, the element through which i was added to A), pick randomly any such that a vertex $i \in N_{i\setminus p(i)} \setminus A$ and add it to A.

(Piv) Repeat (Piii) until $A = \mathcal{V}$.

Given some $i \in A$, we will denote the **descendants** of i, that is, the set of $j \in A$ such that $p(j) = i$, by $D(i)$.

If we take $\mathcal{G}_0$ the graph whose vertices are in the set

$$\{N_{i_0}\} \cup \{N_{i\setminus p(i)}\}_{i\in A\setminus\{i_0\}} \tag{7}$$

and whose edges connect N_{i_0} with $N_{i\setminus p(i)}$ provided $p(i) = i_0$ and $N_{i\setminus p(i)}$ with $N_{k\setminus p(k)}$ provided $p(i) = k$, then, by construction, $\mathcal{G}_0$ is a tree.

One can run a BP algorithm similar to the one proposed by the KCN-method on this graph, where, if we relabel the vertices in Eq. (7) taking i_0 for N_{i_0} and i for $N_{i\setminus p(i)}$, then we can define the set of messages

$$\{m^{(t)}_{i\to p(i)}, m^{(t)}_{p(i)\to i}\}_{i\in A, t\geq 0}, \text{ where}$$
$$m^{(t)}_{i\to p(i)}, m^{(t)}_{p(i)\to i} : X \to \mathbb{R}_{\geq 0}.$$

The messages are uniformly initialized,

$$m^{(t)}_{i\to p(i)}(x_i), m^{(t)}_{p(i)\to i}(x_i) \equiv 1/|X| \text{ for all } x_i \in X, \tag{8}$$

and are updated, for $t \geq 0$, according to the following equations:

$$\begin{aligned} m^{(t+1)}_{i\to p(i)}(x_i) &\equiv \frac{1}{M^{(t+1)}_{i\to p(i)}} \operatorname{tr}\left(S_{i\setminus p(i)} \prod_{k\in D(i)} m^{(t)}_{k\to i} \right) \\ m^{(t+1)}_{p(i)\to i}(x_i) &\equiv \frac{1}{M^{(t+1)}_{p(i)\to i}} \operatorname{tr}\left(S_{p(i)\setminus p(p(i))} m^{(t)}_{p(p(i))\to p(i)} \prod_{k\in D(p(i))\setminus\{i\}} m^{(t)}_{k\to p(i)} \right) \end{aligned} \tag{9}$$

for all $x_i \in X$, where $\{S_{i\setminus p(i)}\}_{i\in A}$ denotes the product of functions included in either $N_{i\setminus p(i)}$, provided $i \in A \setminus \{i_0\}$, or N_{i_0}, if $i = i_0$, and $M^{(t+1)}_{i\to p(i)}$ and $M^{(t+1)}_{p(i)\to i}$ are normalization constants

$$\begin{aligned} M^{(t+1)}_{i\to p(i)} &\equiv \sum_{x_i} \operatorname{tr}\left(S_{i\setminus p(i)} \prod_{k\in D(i)} m^{(t)}_{k\to i} \right) \\ M^{(t+1)}_{p(i)\to i} &\equiv \sum_{x_i} \operatorname{tr}\left(S_{p(i)\setminus p(p(i))} m^{(t)}_{p(p(i))\to p(i)} \prod_{k\in D(p(i))\setminus\{i\}} m^{(t)}_{k\to p(i)} \right) \end{aligned} \tag{10}$$

Once the messages have converged, and denoting them by $\{m_{i\to p(i)}, m_{p(i)\to i}\}_{i\in A\setminus i_0, t\geq 0}$, we can use the following inference equations:

(Ii) To compute the marginals p_i, provided $i \in N_{j\setminus p(j)}$, we use the following equation:

$$p_i(x_i) = \frac{1}{M_i} \operatorname{tr}_{\setminus x_i}\left(S_{j\setminus p(j)} m_{p(j)\to j} \prod_{k\in D(j)} m_{k\to j} \right)$$

for all $x_i \in X$, where M_i is a normalization constant:

$$M_i \equiv \operatorname{tr}\left(S_{j\setminus p(j)} m_{p(j)\to j} \prod_{k\in D(j)} m_{k\to j} \right).$$

(Iii) To compute the internal energy U, we use the following equation:

$$U = -\sum_{a\in\mathcal{F}} \frac{1}{Z_{\partial a}} \operatorname{tr}\left(\log(f_a)\, S_{i_a\setminus p(i_a)} \prod_{k\in D(i_a)} m_{k\to i_a} \right), \tag{11}$$

where $Z_{\partial a}$ is a normalization constant

$$Z_{\partial a} \equiv \operatorname{tr}\left(S_{i_a\setminus p(i_a)} \prod_{k\in D(i_a)} m_{k\to i_a} \right)$$

and $N_{i_a\setminus p(i_a)}$ denotes the neighborhood that f_a belongs to.

(Iiii) The partition function Z can be computed through the following equation:

$$Z = \prod_{a\in A} \operatorname{tr}\left(S_{i_a\setminus p(i_a)} \widehat{m}_{p(i_a)\to i_a} \prod_{k\in D(i_a)} \widehat{m}_{k\to i_a} \right), \quad (12)$$

where $\{\widehat{m}_{i\to p(i)}, \widehat{m}_{p(i)\to i}\}_{i\in A\setminus\{i_0\}, t\geq 0}$ is a rescaling of the messages such that

$$\operatorname{Tr}\left(\widehat{m}_{i\to p(i)}\widehat{m}_{p(i)\to i}\right) = 1. \quad (13)$$

(Iiv) The Shannon entropy S can be computed via the following equation from statistical mechanics:

$$S = \log Z + U. \quad (14)$$

Instead of proving these inference equations now, we will show how to derive similar equations in other contexts (namely, (Ii) and (Iii) can be derived as Eq. (18) and Eq. (20) and (Iiii) as Eq. (33)).

In case we group using a KCN-parameter that is smaller than the largest loop length, we can also associate a graph $\mathcal{G}_0$ (which will not be a tree), and use the same inference equations (adapted for the new topology of the graph in a natural way). In particular, in order to construct this graph, we follow (Pi)-(Piv) and then, instead of using $\{S_{i\setminus p(i)}\}_{i\in A}$ in the update equations, we use $\{S_{i_m\setminus\cup_{n=1}^{m-1} i_n}\}_{m\geq 0}$, where $A = (i_m)_{m\geq 0}$ is a numbering of A according to the order in which they are incorporated following (Pi)-(Piv). Note that, in this instance, some messages will have dimension larger than $|X|$.

To complement this section, we consider another tree-equivalent method in Appendix H.

B. The tree-equivalent method vs the KCN-methods

The tree-equivalent method can explain why the KCN-method is exact only when the largest loop length is bounded by the KCN-parameter. Moreover, in Appendix VI C we discuss how the relation between the tree-equivalent and the KCN-method can provide some justification to the improvement on BP given by scheduling.

We can think of the KCN-method as creating one tree $\mathcal{G}_0^i$ following (Pi)-(Piv) using each $i \in \mathcal{G}$ as seed node. In particular, the KCN-method follows (Pi)-(Piv) for each $i \in \mathcal{G}$ except for the fact that the tree is directed, in the sense that the edge between each pairs of nodes in that tree is directed from $N_{j\setminus p(j)}$ to $N_{p(j)\setminus p(p(j))}$. The edge direction means that we only send a single messages along the edge. Once the messages have converged, we average (in some sense) the inference results given by each of them to obtain the quantities of interest.

One may think that Kirkley et al. give preference to their method over the tree-equivalent one since it allows, at least provided the loop length is bounded by the KCN-parameter, to reduce the computational complexity while keeping the accuracy intact. However, this is not necessarily the case (Appendix A):

- when there are only **bounded loops** (i.e. the KCN-parameter bounds the loops lengths), one can find for most graphs $\mathcal{G}$ a seed such that either the tree-equivalent method is less complex than the KCN-method, or vice versa;
- when there are **unbounded loops** (i.e. with length larger than the KCN-parameter), it may be difficult to predict which seed to use in the tree-equivalent method in order to achieve the best trade-off between accuracy and complexity.

In the case of unbounded loops, averaging over all possible trees as the KCN-method does is a sensible choice. In particular, note that the KCN-method does not just run a set of tree-equivalent methods in parallel, but provides a way that can potentially reduce the runtime complexity.

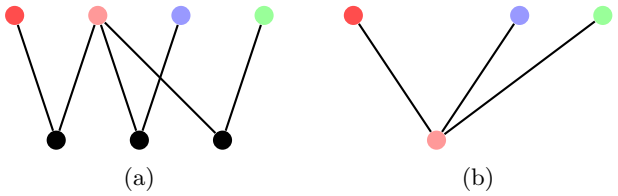
In general, we are interest in the unbounded loop case, such that it feels more natural to use the KCN-method for our analysis. Therefore, in Section IV and Section V, we will first build inference equations for the bounded loop case, and then continue by generalizing it to the unbounded loop case.

IV. BP FOR LOOPY TENSOR NETWORKS

The KCN-method can be adapted to tensor networks. The same holds for the tree-equivalent method for the reasons discussed in Section III B. In Section VI B we show that BlockBP [26, 27] is actually an instance of this method on tensor networks.

Since the variables that compose a tensor network $T = (T_i)_i$ are only shared (at most) by two functions, T admits a simple representation via a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where we associate one node in $\mathcal{V}$ to each function in T and one edge in $\mathcal{E}$ to each variable in T. Moreover, a node is an endpoint of an edge whenever the function associated to the node depends on the variable associated to the edge. We call $\mathcal{G}$ the simplified factor graph associated to T. An example can be found in Figure 3. (Note that, in the following, we assume we are given a tensor network whose variables are shared exactly by two tensors. If there were some variables such that a single tensor depends on them, the so-called dangling edges, these can be naturally incorporated into the scheme.)

Note that, in the case of tensor networks, the normalization constant of the graphical model coincides with the trace of the tensor network $Z = \operatorname{tr}(T)$. Recall, furthermore, that we assume the tensor networks in this work to be connected. (Otherwise, we apply our method to each connected component.)



(a) (b)

FIG. 3: The factor graph of a tensor network (a) and its associated simplified graph (b).

In order to provide a BP algorithm for tensor networks in the spirit of [14], we first consider the case where the maximal loop length ℓ is bounded by the KCN-parameter ℓ_0, $\ell \leq \ell_0$, and then the case where $\ell > \ell_0$. The reason we consider them separately is that, contrary to the case of networks, the update equations will no longer be well defined when the loop bound is not fulfilled and we have to introduce new messages in that scenario.

Before we introduce the update and inference equations, we define some neighborhoods that fulfill a role similar to those in Section II B.

In fact, the definitions in our case can be used to write the update equations directly, simplifying their form.

Given tensor T_i, we define its ℓ_0-tensor-neighborhood $S_i^{\ell_0}$ as T_i together with its nearest tensor-neighbors plus all the tensors that belong to a path of length lower or equal to ℓ_0 that joins two nearest tensor-neighbors to T_i. Since the neighborhood is defined using only tensors (that is, only vertices in the simplified associated graph), and given another tensor T_j, we can define the analogous of the neighborhood difference, namely the ℓ_0-tensor-neighborhood difference $S_{i\setminus j}$, and the related ℓ_0-tensor-neighborhood intersection $S_{i\cap j}^{\ell_0}$, directly via set operations on the tensor-neighborhoods,

$$S_{i\setminus j}^{\ell_0} \equiv S_i^{\ell_0} \setminus S_j^{\ell_0},$$
$$S_{i\cap j}^{\ell_0} \equiv S_i^{\ell_0} \cap S_j^{\ell_0}.$$

As in the case of networks, we drop the ℓ_0 index in the following for simplicity.

A. Bounded loop length

In case the loops' length is bounded by ℓ_0, we can define the BP equations similar to those in [14]. We define the set of messages

$$\{m_{i\to j}^{(t)}\}_{i\in\mathcal{V}, j\in N_i, t\geq 0}, \text{ where}$$
$$m_{i\to j}^{(t)} : X^{|T_{i\setminus j}|} \to \mathbb{R}_{\geq 0}$$

and $T_{i\setminus j}$ denotes the set of variables that T_i depends on and that are shared by some tensor that does not belong to S_j. (In the following, in case $|T_{i\setminus j}| = 0$, we assume $m_{i\to j}$ to have dimension one and to be equal to one for simplicity.)

We initialize the messages uniformly

$$m_{i\to j}^{(0)}(x_{i\setminus j}) \equiv 1/|X|^{|X_{i\setminus j}|} \text{ for all } x_{i\setminus j} \in T_{i\setminus j}, \tag{15}$$

and update them, for $t \geq 0$, according to the following equation:

$$m_{i\to j}^{(t+1)}(x_{i\setminus j}) \equiv \frac{1}{M_{i\to j}^{(t+1)}} \operatorname{tr}\left(S_{i\setminus j} \prod_{k\in S_{i\setminus j}} m_{k\to i}^{(t)}\right), \tag{16}$$

where $M_{i\to j}^{(t+1)}$ is a normalization constant

$$M_{i\to j}^{(t+1)} \equiv \sum_{x_{i\setminus j}} \operatorname{tr}\left(S_{i\setminus j} \prod_{k\in S_{i\setminus j}} m_{k\to i}^{(t)}\right). \tag{17}$$

Note that the approach in [30] corresponds to a weaker version of ours where the neighborhood does not even include the nearest neighbors of the tensor but only the tensor itself.

Given the fact that we group using the bound on the loops ℓ_0, the iteration of the previous equations converges to a fixed point, that is, $m_{i\to j}^{(T+k)} = m_{i\to j}^{(T)}$ for all $k \geq 0$, where T is the diameter of a certain graph that is associated to the grouping procedure. We denote the set of converged messages by eliminating the time label $\{m_{i\to j}\}_{i\in\mathcal{V}, j\in N_i, t\geq 0}$, where $m_{i\to j} = m_{i\to j}^{(T)}$.

Once the messages have converged, we can use them to infer local and global quantities of interest, like marginals, the internal energy, the partition function and the Shannon entropy. In particular, as we show in Appendix B, we have the following equations:

(Ei) The marginal over variable i, p_i, fulfills the following equation:

$$p_i(x) = \frac{1}{2Z_{i_1}} \operatorname{tr}_{\setminus x_i}\left(S_{i_1} \prod_{k\in S_{i_1}} m_{k\to i_1}\right) + \frac{1}{2Z_{i_2}} \operatorname{tr}_{\setminus x_i}\left(S_{i_2} \prod_{k\in S_{i_2}} m_{k\to i_2}\right) \text{ for all } x \in X, \tag{18}$$

where T_{i_1} and T_{i_2} are the tensors that depend on variable i, and Z_{i_1} and Z_{i_2} are, respectively, normalization constants

$$Z_{i_j} \equiv \mathrm{tr}\left(S_{i_j} \prod_{k \in S_{i_j}} m_{k \to i_j}\right) \tag{19}$$

for $j = 1, 2$.

(Eii) In order to compute the internal energy U, we treat the tensor network as if it were encoding a Boltzmann distribution. In particular, if we denote by ∂T_i the set of variables that tensor T_i depends on, and we associate to the tensor network a set of energy functions $\{E_i\}_{i \in \mathcal{G}}$, where

$$E_i : X^{|\partial T_i|} \to \mathbb{R}_{\geq 0}, \text{ and}$$
$$E_i(x_{\partial T_i}) \equiv -\log T_i(x_{\partial T_i}) \text{ for all } x_{\partial T_i} \in \partial T_i,$$

then we get that

$$U = -\sum_{i \in \mathcal{V}} \frac{1}{Z_{\partial T_i}} \mathrm{tr}\left(\log\left(T_i\right) S_i \prod_{k \in S_i} m_{k \to i}\right), \tag{20}$$

where $Z_{\partial T_i}$ is a normalization constant

$$Z_{\partial T_i} \equiv \mathrm{tr}\left(S_i \prod_{k \in S_i} m_{k \to i}\right). \tag{21}$$

(Eiii) The partition function $Z = \mathrm{tr}(T)$ can be computed via the following equation:

$$Z = \frac{\prod_{i \in \mathcal{V}} \mathrm{tr}\left(S_i \prod_{j \in S_i} m_{j \to i}\right)}{\prod_{((i,j)) \in \mathcal{G}} \mathrm{tr}\left(S_{i \cap j} m_{j \to i} \prod_{k \in S_{i \cap j}} m_{k \to j}\right)^{\frac{2}{|S_{i \cap j}|}}}, \tag{22}$$

where $((i,j)) \in \mathcal{G}$ denotes the set of pairs of tensors T_i, T_j such that $T_i \in S_j$ and $T_j \in S_i$.

(Eiv) Lastly, we can compute the Shannon entropy S via Eq. (14).

B. Unbounded loop length

In this case, the inference equations are well defined, but we may find instances where using the update equations directly may result in a contraction over tensor networks with missing legs. Figure 4 illustrates an example of this situation.

The missing legs are not an issue in the network case since all messages are vectors of dimension $|X|$. For tensor networks, however, Eq. (16) is not well defined because the missing legs. In order for us to compute $m_{i \to j}^{(t+1)}$ for some $t \geq 0$, we need to contract $S_{i \setminus j}$ with $m_{k \to j}^{(t)}$ for all $T_k \in S_{i \setminus j}$. However, $S_{i \setminus j}$ may contain tensors that share some variables with tensors that belong to $S_{i \cap j} \setminus \{S_i\}$. Hence, by definition, these edges are not shared by $m_{k \to i}^{(t)}$ for any $k \in S_{i \setminus j}$ and the contraction in Eq. (16) is not well defined.

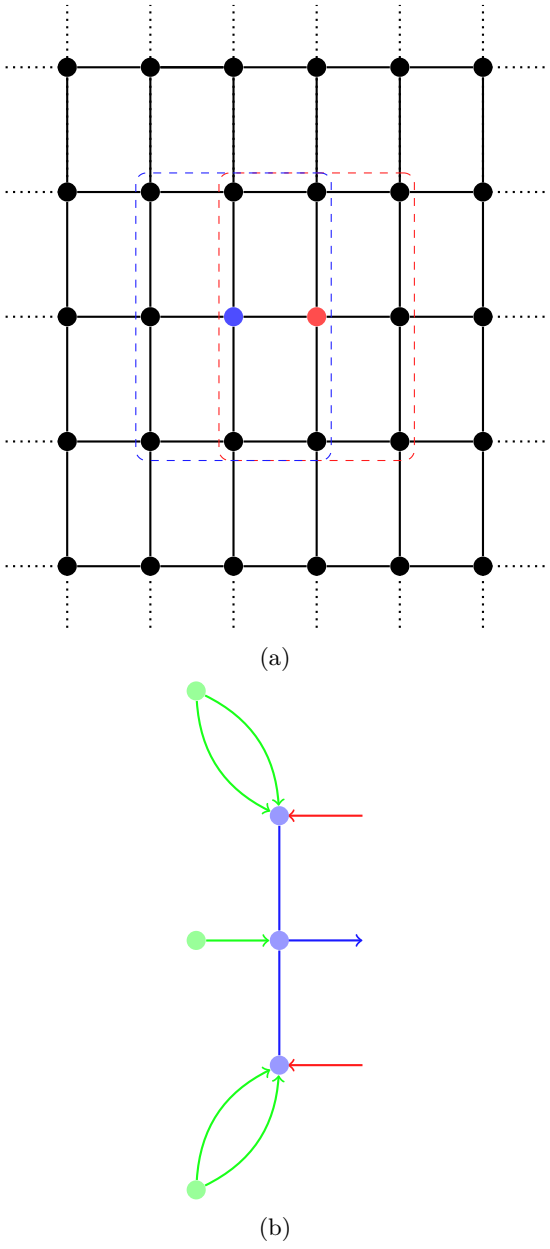


FIG. 4: A subgraph of the lattice (a). The colored dashed lines enclose the N_i^4 neighborhoods for the node i of the same color. Using directly the update equations in the KCN-method fails, since the message (b) from the blue node to the red node is not well-defined, as it does not contract over the edges in red.

To solve the issue with the missing legs and the update of $m_{i \to j}^{(t+1)}$ for $t \geq 0$, we will add some more messages. Before we do this formally, we first note that, given $T_k \in S_{i \setminus j}$, we have, for each variable v that T_k

depends on, that v connects T_k to a tensor that is either in (i) $S_{i\setminus j}$, (ii) $S_{i\cap j}$ or (iii) $T \setminus S_i$. Hence, since the connections between tensor within $S_{i\setminus j}$ are internal to $S_{i\setminus j}$ and the connections to tensors in $T \setminus S_i$ are those in $\{m^{(t)}_{k\to i}\}_{k\in S_{i\setminus j}}$, it is sufficient to add a message $m^{(t)}_{i\cap j\to i}$ with the dimension of the variables that connect tensors in $S_{i\setminus j}$ to tensors in $S_{i\cap j}\setminus\{S_i\}$. The new message $m^{(t)}_{i\cap j\to i}$ also needs to be updated. Reasoning analogously, we can compute $m^{(t+1)}_{i\cap j\to i}$ by contracting the tensors in $S_{i\cap j}$ with $m^{(t)}_{i\to j}$ and, for each $T_k \in S_{i\cap j} \setminus \{S_i\}$, with $m^{(t)}_{k\to i}$.

Formally, whenever the KCN-parameter is smaller than length of the largest loops, then we define the set of messages

$$\begin{aligned}&\{m^{(t)}_{i\to j}, m^{(t)}_{i\cap j\to i}\}_{i\in\mathcal{V}, j\in N_i, t\geq 0}, \text{ where}\\ &m^{(t)}_{i\to j} : X^{|T_{i\setminus j}|} \to \mathbb{R}_{\geq 0} \text{ and}\\ &m^{(t)}_{i\cap j\to i} : X^{|X_{i\cap j\to i}|} \to \mathbb{R}_{\geq 0},\end{aligned}$$

with $T_{i\setminus j}$ having the same meaning as in Section IV A and $X_{i\cap j\to i}$ denoting the set of variables that connect tensors in $S_{i\setminus j}$ to tensors in $S_{i\cap j}\setminus\{S_i\}$. (In the following, in case $|X_{i\setminus j}| = 0$ or $|X_{i\cap j\to i}| = 0$, we assume $m^{(t)}_{i\to j}$ or $m^{(t)}_{i\cap j\to i}$, respectively, to have dimension one and to be equal to one for all $t \geq 0$.)

We initialize the messages uniformly

$$\begin{cases} m^{(0)}_{i\to j}(x_{i\setminus j}) \equiv 1/|X|^{|T_{i\setminus j}|} \\ \text{for all } x_{i\setminus j} \in T_{i\setminus j}, \end{cases}$$

$$\begin{cases} m^{(0)}_{i\cap j\to i}(x_{i\cap j\to i}) \equiv 1/|X|^{|X_{i\cap j\to i}|} \\ \text{for all } x_{i\cap j\to i} \in X_{i\cap j\to i}, \end{cases}$$

and update them, for $t \geq 0$, according to the following equations:

$$\begin{cases} m^{(t+1)}_{i\to j}(x_{i\setminus j}) \equiv \frac{1}{M^{(t+1)}_{i\to j}} \operatorname{tr}\left(S_{i\setminus j} m^{(t)}_{i\cap j\to i} \prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}\right) \\ \text{for all } x_{i\setminus j} \in T_{i\setminus j}, \end{cases}$$

$$\begin{cases} m^{(t+1)}_{i\cap j\to i}(x_{i\cap j\to i}) \equiv \frac{1}{M^{(t+1)}_{i\cap j\to i}} \operatorname{tr}\left(S_{i\cap j} m^{(t)}_{i\to j} \prod_{k\in S_{i\cap j}} m^{(t)}_{k\to i}\right) \\ \text{for all } x_{i\cap j\to j} \in X_{i\cap j\to j}, \end{cases} \tag{23}$$

where $M^{(t+1)}_{i\to j}$ and $M^{(t+1)}_{i\cap j\to i}$ are normalization constants

$$\begin{cases} M^{(t+1)}_{i\to j} \equiv \sum_{x_{i\setminus j}} \operatorname{tr}\left(S_{i\setminus j} m^{(t)}_{i\cap j\to i} \prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}\right), \\ M^{(t+1)}_{i\cap j\to j} \equiv \sum_{x_{i\cap j\to i}} \operatorname{tr}\left(S_{i\cap j} m^{(t)}_{i\to j} \prod_{k\in S_{i\cap j}} m^{(t)}_{k\to i}\right). \end{cases} \tag{24}$$

Note that Eq. (23) coincides with Eq. (16) provided $|X_{i\cap j\to i}| = 0$, which is exactly the situation whenever the loop length is bounded by the KCN-parameter.

An example of how we can solve the issues regarding the update of the messages presented in Figure 4 using the new messages can be found in Figures 5 and 6. Note that the only two types of messages that we include are sufficient since, by symmetry, the rest are essentially the same.

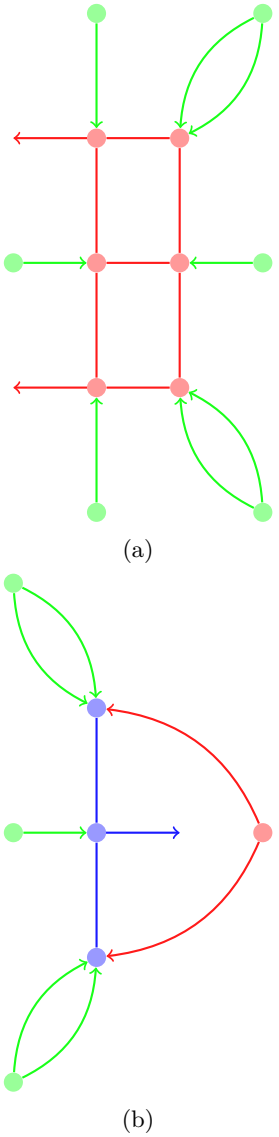
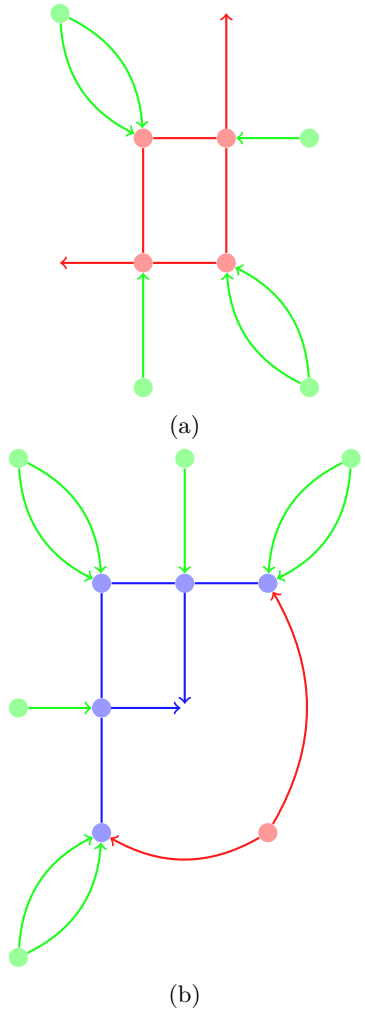


FIG. 5: One sort of intersection message (a) and the message update it enables (b). By symmetry, this figure together with Figure 6 cover all cases.

As a last remark, note that the only issue when the KCN-parameter is smaller than the maximal loop length takes place in the update equations. Once we have converged (or run out of iterations) we can use the same inference equations (Ei)-(Eiv) as if the the maximal loop length was bounded by the KCN-parameter.

The only issue is that, if that is not the case, then the inference equations will no longer be exact, but approximate. The use of the same inference equations (up to some minor modifications, see Appendix B), however, is common in the BP field and goes back to the Bethe



(a)

(b)

FIG. 6: One sort of intersection message (a) and the message update it enables (b). By symmetry, this figure together with Figure 5 cover all cases.

approximation [1, 34]. (Note that the Bethe approximation essentially consists in overcounting the information in certain variables and then subtracting what has been overcounted [1], which is what we do when we infer the partition function and entropy, the difference being that, in our case, we overcount not only variables but also functions.)

In some instances, it may be possible to find less complex BP update rules by using some symmetries in the graph. This is the case for the lattice, as we discuss in Appendix C.

V. BP FOR LOOPY GRAPHICAL MODELS

The KCN-method can be adapted to arbitrary graphical models, too. Similarly, once more, the same holds for the tree-equivalent method for the reasons discussed in Section III B.

The intuition behind our extension is to treat variables and factors in the same way when defining neighborhoods and to only distinguish them when introducing the messages. At this stage we will treat factors in the spirit of tensors from the tensor network approach, and variables in the spirit of variables from the network approach.

To begin with, take a graphical model P and consider its associated factor graph $\mathcal{G} = (\mathcal{F} \cup \mathcal{V}, \mathcal{E})$. Following our approach to tensor networks, for each vertex $i \in \mathcal{F} \cup \mathcal{V}$, we create a neighborhood S_i as before and define $S_{i\setminus j}$ and $S_{i\cap j}$ in the same way. We treat the factor graph of a general graphical model as the simplified factor graph of a tensor network.

A. Bounded loop length

In case the loops in $\mathcal{G}$ are bounded by the KCN-parameter, we define the set of messages

$$\begin{aligned} &\{m^{(t)}_{i\to j}\}_{i\in\mathcal{F}\cup\mathcal{V}, j\in N_i, t\geq 0}, \text{ where} \\ &m^{(t)}_{i\to j} : X^{|X_{i\setminus j}|} \to \mathbb{R}_{\geq 0} \text{ if } i \in \mathcal{F} \text{ and} \\ &m^{(t)}_{i\to j} : X \to \mathbb{R}_{\geq 0} \text{ if } i \in \mathcal{V}, \end{aligned} \tag{25}$$

with $X_{i\setminus j}$ denoting the set of variables that factor i depends on and that do not belong to to S_j.

We initialize the messages uniformly

$$\begin{cases} m^{(0)}_{i\to j}(x_i) \equiv 1/|X| \text{ for all } x_i \in X \text{ if } i \in \mathcal{V}, \\ m^{(0)}_{i\to j}(x_{i\setminus j}) \equiv 1/|X|^{|X_{i\setminus j}|} \text{ for all } x_{i\setminus j} \in X_{i\setminus j} \text{ if } i \in \mathcal{F}, \end{cases} \tag{26}$$

and update them, for $t \geq 0$, according to the following equations:

$$\begin{cases} m^{(t+1)}_{i\to j}(x_i) \equiv \frac{1}{M^{(t+1)}_{i\to j}} \operatorname{tr}_{\setminus x_i}(S_{i\setminus j} \prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}) \\ \text{for all } x_i \in X \text{ if } i \in \mathcal{V}, \\ \\ m^{(t+1)}_{i\to j}(x_{i\setminus j}) \equiv \frac{1}{M^{(t+1)}_{i\to j}} \operatorname{tr}_{\setminus x_{i\setminus j}}(S_{i\setminus j} \prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}) \\ \text{for all } x_{i\setminus j} \in X_{i\setminus j} \text{ if } i \in \mathcal{F}, \end{cases} \tag{27}$$

where $M^{(t+1)}_{i\to j}$ are normalization constants defined as before.

Once the messages have converged, we can take the set of converged messages $\{m_{i\to j}\}_{i\in\mathcal{F}\cup\mathcal{V}, j\in N_i}$ and use them for inference purposes as in Section IV. For instance, we can compute the marginal p_i for each $i \in \mathcal{V}$ via the following equation:

$$\begin{aligned} p_i(x_i) = &\frac{1}{|\partial i|+1}\left(\frac{1}{Z_i}\operatorname{tr}_{\setminus x_i}\left(S_i \prod_{k\in S_i} m_{k\to i}\right)\right) \\ &+ \frac{1}{|\partial i|+1}\left(\sum_{j=1}^{|\partial i|}\frac{1}{Z_{i_j}}\operatorname{tr}_{\setminus x_i}\left(S_{i_j}\prod_{k\in S_{i_j}} m_{k\to i_j}\right)\right), \end{aligned} \tag{28}$$

where $\{i_j\}_{j=1,\dots,|\partial i|}$ are the factors that depend on variable x_i, and Z_i and Z_{i_j} are appropriate normalization constants.

As in the case of Eq. (18) and thinking of the case where the KCN-parameter does not bound the loop length, we again give preference to Eq. (28) since it averages over the other inference equations for p_i.

Assuming the graphical model represents a Boltzmann distribution as in Section IV, we obtain the following equation for the internal energy:

$$U = -\sum_{i\in\mathcal{F}} \mathrm{tr}\left(\log\left(T_i\right)\frac{1}{|\partial T_i|+1}\left(\frac{1}{Z_{\partial T_i}}\,\mathrm{tr}_{\setminus\partial T_i}\left(S_i\prod_{k\in S_i} m_{k\to i}\right)+\sum_{j\in\partial T_i}\frac{1}{Z_j}\,\mathrm{tr}_{\setminus T_i}\left(S_j\prod_{k\in S_j} m_{k\to j}\right)\right)\right), \tag{29}$$

where $Z_{\partial T_i}$ and Z_j for each $j\in\partial T_i$ are appropriate normalization constants.

Regarding the partition function and Shannon entropy, we can use the same formulas as for the tensor network case, namely Eq. (22) and Eq. (14), with the set of neighborhoods that corresponds to an arbitrary graphical model and a slight modification. In particular, for the partition function, we take $S_{i\cap j}$ to be the set of factors within $N_{i\setminus j}$ while $|S_{i\cap j}|$ counts **all** vertices within $S_{i\cap j}$. For Shannon entropy, we use the modified internal energy and partition function.

B. Unbounded loop length

Whenever the loop length is not bounded by the KCN-parameter, we again take care of the missing legs issue by adding intersection messages. In this case, we define the set of messages

$$\begin{aligned}&\{m^{(t)}_{i\to j}, m^{(t)}_{i\cap j\to i}\}_{i\in\mathcal{F}\cup\mathcal{V}, j\in N_i, t\geq 0},\ \text{where}\\ &\{m^{(t)}_{i\to j}\}_{i\in\mathcal{F}\cup\mathcal{V}, j\in N_i, t\geq 0}\ \text{is defined as in } \textit{Eq.}\ (25)\ \text{and}\\ &m^{(t)}_{i\cap j\to i}: X^{|X_{i\cap j\to i}|}\to\mathbb{R}_{\geq 0},\end{aligned} \tag{30}$$

with $X_{i\cap j\to i}$ denoting the set of variables in $S_{i\cap j}\setminus\{i\}$ that are connected to functions in $S_{i\setminus j}$. (In the following, in case $|X_{i\cap j\to i}|=0$, we assume $m^{(t)}_{i\cap j\to i}$ to have dimension one and to be equal to one for all $t\geq 0$.)

We initialize the messages uniformly following Eq. (26) and

$$m^{(0)}_{i\cap j\to i}(x_{i\cap j\to i}) \equiv 1/|X|^{|X_{i\cap j}|}\ \text{for all}\ x_{i\cap j\to i}\in X_{i\cap j\to i}, \tag{31}$$

and update them, for $t\geq 0$, according to the following equations:

$$\begin{cases} m^{(t+1)}_{i\to j}(x_i) \equiv \frac{1}{M^{(t+1)}_{i\to j}}\,\mathrm{tr}_{\setminus x_i}(S_{i\setminus j} m^{(t)}_{i\cap j\to i}\prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}) \\ \text{for all } x_i\in X \text{ if } i\in\mathcal{V}, \\ \\ m^{(t+1)}_{i\to j}(x_{i\setminus j}) \equiv \frac{1}{M^{(t+1)}_{i\to j}}\,\mathrm{tr}_{\setminus x_{i\setminus j}}(S_{i\setminus j} m^{(t)}_{i\cap j\to i}\prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}) \\ \text{for all } x_{i\setminus j}\in X_{i\setminus j} \text{ if } i\in\mathcal{F}, \\ \\ m^{(t+1)}_{i\cap j\to i}(x_{i\cap j\to i}) \equiv \frac{1}{M^{(t+1)}_{i\cap j\to i}}\,\mathrm{tr}_{\setminus x_{i\cap j\to i}}(S_{i\cap j} m^{(t)}_{i\to j}\times \\ \prod_{k\in S_{i\setminus j}} m^{(t)}_{k\to i}) \\ \text{for all } x_{i\cap j\to i}\in X_{i\cap j\to i}, \end{cases} \tag{32}$$

where $M^{(t+1)}_{i\to j}$ and $M^{(t+1)}_{i\cap j\to i}$ are appropriate normalization constants.

After convergence (or after we run out of iterations), we use the inference equations for the case where the KCN-parameter bounds the loop length.

To conclude, note that, in case the graphical model is a network (tensor network), the update equations in Eq. (32) reduce to Eq. (5) (Eq. (23)). Arguing along the same lines of Appendix D, translating an arbitrary graphical model to an equivalent network (or a tensor network) and then using the BP methods defined for them is in general suboptimal regarding complexity.

VI. UNIFYING BP FOR LOOPY TENSOR NETWORKS

In this section, we discuss different approaches to BP on tensor networks. In particular, we focus on two approaches which are related one to each other: tensor network message passing [28], a variation of the KCN-method, and block BP [26, 27], an instance of the tree-equivalent approach. Appendix D discusses whether one could simply map a tensor network into an equivalent network and apply the KCN-method directly. To this end, we consider the association of an equivalent simpler graphical model to another one.

A. Tensor network message passing is the KCN-method with different neighborhoods

The closer to our approach is the one by Wang, Zhang, Pan and Zhang [28], where they provide a modification of the algorithm by Kirkley at al. [14] that applies to certain tensor networks associated with statistical mechanics. In particular, they start from a network (just like [14] does) and then turn it into a tensor network. The algorithm they present, which they call **tensor network message passing**, is then tailored to these specific types of tensor networks. In particular, since the method essentially passes messages between variables the way [14] does, the issue with the missing leg does not occur. This prevents their method from applying to general tensor networks. In contrast, this does not occur for our method.

A second shortcoming in [28] is that no equation to infer a global quantity, as is provided. The (local) inference equations for they derive can be found in [28, Equations (S7) and (S8)]. We will devote the rest of this section clarify the relation between the method by Wang et al. and that by Kirkley et al. and, as a result, we will show that one can use the inference equations we derived for the KCN-method also for the Wang method, which we refer to as the **WZPZ-method** or WZPZ-approach. This implies, that we make the WZPZ-method available for quantum degenerate maximum likelihood decoding.

The key difference between the work by Wang et al. [28] and that by Kirkley et al. [14] is the definition of the neighborhoods. Although [28] introduces neighborhood in the context of tensor networks, there is no difference if we do so in the context of networks, which we can then directly compare to Kirkley. In the following, we will refer to the neighborhoods in the KCN-method as KCN-neighborhood (and denote them by $N_i^{\ell,K}$) and to those in the WZPZ-method as WZPZ-neighborhoods.

Given the simplified graph of a network $\mathcal{G}$, we define WZPZ-neighborhoods using a parameter that plays a role similar to that of the KCN-parameter ℓ_0 in the previous discussion. Given a subset of the network $N \subseteq \mathcal{G}$, the outer distance of N, $\min_{(a,b)} d_{a,b}(\partial N)$, is the length of the smallest path outside of N that connects two variables in the boundary of N, $a, b \in \partial N$. (The boundary of N refers to variables that have a function that depends on them and is not contained in N.)

Given some variable $i \in \mathcal{G}$ and a parameter ℓ_0, its WZPZ-neighborhood N_i^{W,ℓ_0} is constructed by incorporating variables and functions into N_i^{W,ℓ_0} until $\min_{(a,b)} d_{a,b}(\partial N_i^{W,\ell_0}) \geq \ell_0$. More specifically [28], one can construct N_i^{W,ℓ_0} by recursively adding variables and edges as follows:

(Ri) Add i as well as all the edges connected to it and the variables they connect it to.

(Rii) For each $\ell \leq \ell_0$ starting from $\ell = 1$, add all variables and edges along paths of length ℓ or less that connect a pair of variables in the boundary N_i^{W,ℓ_0}. Repeat the procedure until neither edges nor variables can be added anymore. After that, start the step again, this time with $\ell+1$ (provided $\ell+1 \leq \ell_0$).

We refer to the iteration at which each node v (and analogously for each edge) is incorporated to N_i^{W,ℓ_0} as its **generation** and denote it by $g_i(v)$ or, whenever i is clear, $g(v)$.

Although they use KCN-neighborhoods, both the approach by Kirkley et al. to networks and our approach to tensor networks can analogously be applied using WZPZ-neighborhoods or, respectively, their natural extension to tensor networks. Moreover, we can use the same inference equations. To see this, let us argue as in the context KCN-neighborhoods.

We start by assuming that the neighborhoods we use $\{N_i^{W,\ell_0}\}_{i\in\mathcal{V}}$ take into account all the correlations in the graph, $\min_{(a,b)} d_{a,b}(\partial N_i^{W,\ell_0}) = \infty$. If that is the case, then one can show (see the proof of (Wvii) in Appendix E 1) that there exists some $t_{\ell_0} \in \mathbb{N}$ such that

$$N_i^{W,\ell_0} = N_i^{K,t_{\ell_0}}$$

for all $i \in \mathcal{V}$. Hence, one is under the assumptions where one can derive exact inference equations via KCN-neighborhoods and, since the neighborhoods are equal, the same equations also hold for WZPZ-neighborhoods. This allows us to go beyond [28] and provide inference equations for global quantities. If the assumption that $\min_{(a,b)} d_{a,b}(\partial N_i^{W,\ell_0}) = \infty$ fails, we can again use the equations derived assuming it holds and do approximate inference using them (see Appendix E 2).

We consider the general relation between WZPZ- and KCN-neighborhoods in Appendix E 1.

B. BlockBP is the tree-equivalent approach

Following the translation of standard BP to tensor networks [30], BlockBP was recently introduced [26, 27] in the spirit of the tree-equivalent method but in the context of tensor networks.

In Block-BP, a PEPS tensor network [26, 31, 32] is partitioned into blocks of square shape and messages are exchanged between blocks in the spirit of the tree-equivalent method. Hence, at the expanse of fully contracting within the blocks (although approximate contraction methods like bMPS have also been considered [26, 27]), the correlations within the blocks can be computed to larger accuracy, hence improving the performance compared to simply doing BP directly.

We can think of Block-BP (and of the tree-equivalent methods in general) as a limit case of our extension of the KCN-approach to tensor networks, in the sense that only variables belong to the intersection between different neighborhoods. Hence, although the inference equations from Section IV may not be well-defined, they can be naturally extended to this limit case. As an instance

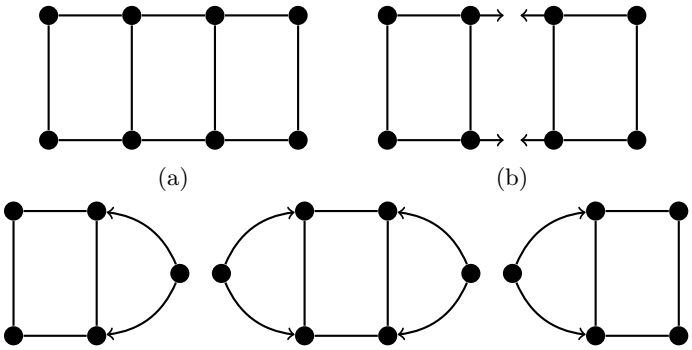


FIG. 7: An improvement on the BlockBP algorithm when using a Kirkley-like algorithm. (a) Original graph, (b) an instance of BlockBP, (c) an instance of a KCN-like algorithm.

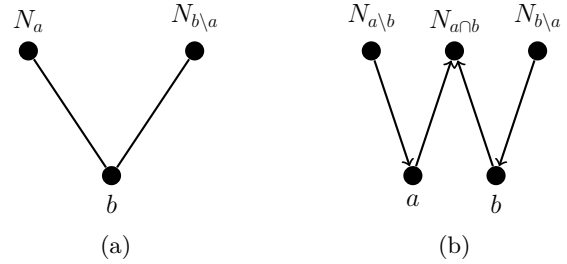


FIG. 8: The Tanner graph of the tree-equivalent method (a – i.e. BP without scheduling) and the KCN-method (b – i.e. BP with scheduling).

of this, we can consider the inference equation for the partition function used in [27]:

$$Z = \prod_{i\in\mathcal{G}} \mathrm{tr}\left(S_i \prod_{j\in\mathcal{NN}(i)} \widehat{m}_{j\to i}\right), \tag{33}$$

where $\{S_i\}_{i\in\mathcal{G}}$ are the blocks used in BlockBP, $\mathcal{NN}(i)$ refers to the blocks that share variables with S_i, $\{m_{i\to j}\}_{i\in\mathcal{G},j\in\mathcal{NN}(i)}$ are the BP messages between blocks, and $\{\widehat{m}_{i\to j}\}_{i\in\mathcal{G},j\in\mathcal{NN}(i)}$ are the rescaled messages that fulfill

$$\mathrm{tr}\left(\widehat{m}_{i\to j}\widehat{m}_{j\to i}\right) = 1. \tag{34}$$

for all $i \in \mathcal{G}, j \in \mathcal{NN}(i)$.

We can interpret Eq. (33) as a simplification of Eq. (22), where the only redundant information shared between blocks is in the variables that connect them. Similarly to how we avoid overcounting in Eq. (22) by subtracting the overlap between different tensor-neighborhoods, we avoid it here by normalizing the messages along the indexes shared between blocks S_i and S_j. To clarify what we mean by this, in Appendix F, we derive Eq. (33) along the lines of our derivation of Eq. (22).

Although one can improve on BlockBP in some instances where the algorithm is exact (see Figure 7 for example), it is in general not possible to do so when the blocks form a two-dimensional tree. We discuss the relation between the two methods whenever BlockBP is not exact in Section VII.

C. Application: Improving BP through scheduling

Scheduling is a variation of the original BP scheme where messages are not sent through all the edges of the Tanner graph at every time step (flooding). Instead, there is a schedule that determines which parts of the Tanner graph exchange information with others at every time step. Despite it having been reported in some instances [35, 36], the improvement provided by scheduling is not well understood.

Herein, we argue that the KCN-method is a theoretical construction that profits from scheduling in order to improve the performance of BP. For simplicity, we do so in the context of networks, although we could argue analogously for any graphical model.

Whenever the KCN-parameter bounds the loop length, we can think of the KCN-method as the introduction of a scheduling into the tree-equivalent method, where we achieve exact results (as we would without scheduling) using less computational power and, hence, we improve on the performance of BP. In order to achieve the improvement, one needs to pick the seed for the tree-equivalent method following the proof of (Tii) in Appendix A. In the following, when we refer to the **tree-equivalent** method, we assume that the seed has been picked in such a way.

In order to understand how the scheduling is related to the improvement, let us consider a graph $\mathcal{G}$ such that $\mathcal{G} = N_a \cup N_b$ for $a, b \in \mathcal{V}$ and some KCN-parameter ℓ_0. Taking a as seed in the tree-equivalent method, then $m_{a\to b}$ communicates all the information in N_a to $N_{b\setminus a}$ and $m_{b\to a}$ communicates all the information in $N_{b\setminus a}$ to N_a. This corresponds to flooding. On the other hand, the KCN-method profits from the fact that $\mathcal{G} = N_{a\cap b} \cup N_{a\setminus b} \cup N_{b\setminus a}$ to get $m_{a\to b}$ to communicate all the information in $N_{a\setminus b}$ to $N_{a\cap b}$ and $m_{b\to a}$ to communicate all the information in $N_{b\setminus a}$ to $N_{a\cap b}$. Since it is only after the update process has finished, i.e. at the inference stage, that the information in $N_{b\setminus a}$ reaches N_a and the information in $N_{a\setminus b}$ reaches N_b, we can think of the KCN-method as the tree-equivalent method with scheduling and, since it achieves accurate results with less complexity, it provides an instance where the improvement given by scheduling can be clearly explained. The Tanner graph for these tree-equivalent and KCN-instances can be found in Figure 8.

VII. CONCLUSION

We have considered the tree-equivalent and KCN-methods in the context of networks, tensor networks and graphical models in general, deriving BP schemes and providing inference equations. As a result, we have developed a unified view regarding the algorithms that have been proposed in order to make generalized BP explicit. As concrete links, we related BlockBP to the KCN-method by introducing the tree-equivalent approach. Moreover, aside from showing how to extend it to arbitrary tensor networks, we have shown that the WZPZ-approach essentially relies on the same approximation as the KCN-method, allowing us to make it available for degenerate quantum maximum likelihood decoding.

Several question remain to be answered. From a theoretical point of view, it would be interesting to understand better how BP works whenever the graph (or multiple graphs in case of the KCN- or WZPZ-approach) that we construct by gathering nodes together is not a tree. In particular, it would be key to obtain some guidelines regarding how the exchange between complexity and accuracy works depending on how the grouping is done. This ultimately boils down to gaining insight regarding the Bethe approximation, which remains to be properly understood. It would also be interesting to continue along the lines of using symmetry to reduce complexity in the KCN-approach. More specifically, it would be meaningful to provide results regarding under what symmetry conditions (and by how much) one can reduce the complexity while not getting a significant decrease in accuracy.

From an applied point of view, it would be interesting to use the principled approach given by the KCN-method, together with the numerical evidence coming from the usual BP on loopy graphs, as a guideline to constructing codes where the decoding through this method would seem to give good results and then testing it. Regarding previous approaches, it would be interesting to compare our tensor network methods with BlockBP. In particular, to check if one can obtain a similar accuracy than BlockBP by using less computational resources. In order to do so, one could pick any instance of BlockBP, which reduces to picking some block size, and then run our tensor network approach with these blocks as neighborhoods. In this scenario, our tensor network method is still well defined and it would require less space and time complexity and, hence, it would be interesting to check how the accuracy in the partition function varies. It should be noted that, even if the accuracy reduces, this does not necessarily mean that the performance of the tensor network decoder must decrease, since as long as the maximal computed partition functions among the logical cosets coincides with the actual maximal one [3, 27], we will be able to successfully do degenerate quantum maximum likelihood decoding. For such an implementation, let us note that, just like BlockBP reduces complexity by using the MPS-MPO approximation [26, 27], we can use a sweep line algorithm [24] provided the tensor network is 2D or follow a Markov chain Monte Carlo importance sampling approach [14, Section II.C]. Lastly, it would be interesting to see if one can exploit the connection we pointed out between scheduling and the KCN-method in order to obtain improvements on BP more in general.

ACKNOWLEDGEMENTS

P. Hack acknowledges funding by the Munich Quantum Valley, section K7. The research is part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus.

This research was developed in part with funding from the Defense Advanced Research Projects Agency [under the Quantum Benchmarking (QB) program under award no. HR00112230006 and HR001121S0026 contracts]. The views, opinions and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

[1] M. Mezard and A. Montanari, *Information, physics, and computation* (Oxford University Press, 2009).
[2] D. Poulin and Y. Chung, On the iterative decoding of sparse quantum codes, arXiv preprint arXiv:0801.1241 (2008).
[3] S. Bravyi, M. Suchara, and A. Vargo, Efficient algorithms for maximum likelihood decoding in the surface code, Physical Review A **90**, 032326 (2014).
[4] P. Iyer and D. Poulin, Hardness of decoding quantum stabilizer codes, IEEE Transactions on Information Theory **61**, 5209 (2015).
[5] C. Cao and B. Lackey, Quantum lego: Building quantum error correction codes from tensor networks, PRX Quantum **3**, 020332 (2022).
[6] A. de Marti i Olius, P. Fuentes, R. Orús, P. M. Crespo, and J. E. Martinez, Decoding algorithms for surface codes, arXiv preprint arXiv:2307.14989 (2023).
[7] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference* (Morgan kaufmann, 1988).
[8] B. J. Frey and D. MacKay, A revolution: Belief propagation in graphs with cycles, Advances in neural information processing systems **10** (1997).
[9] J. S. Yedidia, W. Freeman, and Y. Weiss, Generalized belief propagation, Advances in neural information processing systems **13** (2000).

[10] J. S. Yedidia, W. T. Freeman, and Y. Weiss, Bethe free energy, kikuchi approximations, and belief propagation algorithms, Advances in neural information processing systems **13** (2001).

[11] J. S. Yedidia, W. T. Freeman, and Y. Weiss, Characterization of belief propagation and its generalizations, IT-IEEE **51**, 2282 (2001).

[12] J. S. Yedidia, W. T. Freeman, Y. Weiss, *et al.*, Understanding belief propagation and its generalizations, Exploring artificial intelligence in the new millennium **8**, 0018 (2003).

[13] J. Old and M. Rispler, Generalized belief propagation algorithms for decoding of surface codes, Quantum **7**, 1037 (2023).

[14] A. Kirkley, G. T. Cantwell, and M. Newman, Belief propagation for networks with loops, Science Advances **7**, eabf1211 (2021).

[15] G. T. Cantwell and M. E. Newman, Message passing on networks with loops, Proceedings of the National Academy of Sciences **116**, 23398 (2019).

[16] A. J. Ferris and D. Poulin, Tensor networks and quantum error correction, Physical Review Letters **113**, 030501 (2014).

[17] T. Farrelly, R. J. Harris, N. A. McMahon, and T. M. Stace, Tensor-network codes, Physical Review Letters **127**, 040507 (2021).

[18] C. T. Chubb and S. T. Flammia, Statistical mechanical models for quantum codes with correlated noise, Annales de l'Institut Henri Poincaré D **8**, 269 (2021).

[19] J. Tindall and M. Fishman, Gauging tensor networks with belief propagation, SciPost Physics **15**, 222 (2023).

[20] J. Tindall, M. Fishman, E. M. Stoudenmire, and D. Sels, Efficient tensor network simulation of IBM's eagle kicked Ising experiment, PRX Quantum **5**, 010308 (2024).

[21] C. Piveteau, C. T. Chubb, and J. M. Renes, Tensor network decoding beyond 2d, arXiv preprint arXiv:2310.10722 (2023).

[22] J. Gray and S. Kourtis, Hyper-optimized tensor network contraction, Quantum **5**, 410 (2021).

[23] S. Sahu and B. Swingle, Efficient tensor network simulation of quantum many-body physics on sparse graphs, arXiv preprint arXiv:2206.04701 (2022).

[24] C. T. Chubb, General tensor network decoding of 2d pauli codes, arXiv preprint arXiv:2101.04125 (2021).

[25] T. Begušić, J. Gray, and G. K.-L. Chan, Fast and converged classical simulations of evidence for the utility of quantum computing before fault tolerance, Science Advances **10**, eadk4321 (2024).

[26] C. Guo, D. Poletti, and I. Arad, Block belief propagation algorithm for two-dimensional tensor networks, Physical Review B **108**, 125111 (2023).

[27] A. Kaufmann and I. Arad, A blockBP decoder for the surface code, arXiv preprint arXiv:2402.04834 (2024).

[28] Y. Wang, Y. E. Zhang, F. Pan, and P. Zhang, Tensor network message passing, Physical Review Letters **132**, 117401 (2024).

[29] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques* (MIT press, 2009).

[30] R. Alkabetz and I. Arad, Tensor networks contraction and the belief propagation algorithm, Physical Review Research **3**, 023073 (2021).

[31] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Annals of physics **349**, 117 (2014).

[32] J. C. Bridgeman and C. T. Chubb, Hand-waving and interpretive dance: an introductory course on tensor networks, Journal of physics A: Mathematical and theoretical **50**, 223001 (2017).

[33] Y. Weiss, Correctness of local probability propagation in graphical models with loops, Neural computation **12**, 1 (2000).

[34] H. A. Bethe, Statistical theory of superlattices, Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences **150**, 552 (1935).

[35] J. Du Crest, F. Garcia-Herrero, M. Mhalla, V. Savin, and J. Valls, Layered decoding of quantum LDPC codes, in *2023 12th International Symposium on Topics in Coding (ISTC)* (IEEE, 2023) pp. 1–5.

[36] Y.-M. Chang, A. I. V. Casado, M.-C. Chang, and R. D. Wesel, Lower-complexity layered belief-propagation decoding of LDPC codes, in *2008 IEEE International Conference on Communications* (IEEE, 2008) pp. 1155–1160.

[37] D. J. MacKay, A conversation about the bethe free energy and sum-product, Tech. Rep. of Mitsubishi Electric Research Lab. (2001).

[38] F. Eaton and Z. Ghahramani, Model reductions for inference: Generality of pairwise, binary, and planar factor graphs, Neural computation **25**, 1213 (2013).

[39] Y. Weiss and W. T. Freeman, On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs, IEEE Transactions on Information Theory **47**, 736 (2001).

[40] I. Glasser, N. Pancotti, and J. I. Cirac, From probabilistic graphical models to generalized tensor networks for supervised learning, IEEE Access **8**, 68169 (2020).

[41] J. Pearl *et al.*, Models, reasoning and inference, Cambridge, UK: CambridgeUniversityPress **19**, 3 (2000).

[42] R. J. Wilson, *Introduction to graph theory* (Pearson Education India, 1979).

Appendix A: Comparing the tree-equivalent and the KCN-method for networks in terms of complexity

Given a network with simplified factor graph $\mathcal{G}$ and loop length bounded by ℓ_0, the following statements hold:

(Ti) There exists an instance of the tree-equivalent method whose time complexity is smaller than that of the KCN-method if and only if there exists some $i \in \mathcal{G}$ such that $N_i^{\ell_0} \subsetneq \mathcal{G}$.

(Note that by **time complexity** we refer to the complexity of updating the messages until convergence together with that of computing the most complex inference equation.)

To show (Ti), we begin by considering necessity. To prove it, we argue by contrapositive and assume that $N_i^{\ell_0} = \mathcal{G}$ for all $i \in \mathcal{G}$. In this case, the update equations are trivial for both methods and the inference equations use the same neighborhoods and, hence, are equally complex. Hence, all instances of the tree-equivalent method are equivalent to the KCN-method regarding time complexity.

We conclude by showing sufficiency. We split the proof in three different cases:

(I) There exists some $v_0 \in \mathcal{G}$ such that $N_{v_0}^{\ell_0} = \mathcal{G}$. In this case, denoting by $i_0 \in \mathcal{G}$ some vertex such that $N_{i_0}^{\ell_0} \subsetneq \mathcal{G}$, the tree method that uses i_0 as seed is faster than the KCN-method. In particular, the time complexity of the tree-equivalent method is, essentially, twice (one for updating and one for inferring) that of the maximum complexity of summing over N_{i_0} or $N_{v_0 \setminus i_0}$. In contrast, the complexity of the KCN-method is, at least, the sum of that of summing over $N_{v_0 \setminus i_0}$ (update complexity) and that of summing over N_{v_0} (inference complexity). One can then consider separately the cases where summing over N_{i_0} is more complex than doing so over $N_{v_0 \setminus i_0}$ and its opposite, and conclude the tree-equivalent method is less complex.

(II) For all $i \in \mathcal{G}$, $N_i^{\ell_0} \subsetneq \mathcal{G}$ and there is no instance of (Pi)-(Piv) such that $p(p(i))$ exists. This case cannot happen, as we will show by reduction to the absurd. If this case was possible, then let us pick a vertex $a \in \mathcal{G}$. Since $N_a^{\ell_0} \subsetneq \mathcal{G}$, there exists some vertex $b \in N_a^{\ell_0}$ such that there exists a vertex $c \in N_{b \setminus a}^{\ell_0}$. This leads to a contradiction, since, if we take a as seed and follow $(Pi)-(Piv)$, then $p(p(c)) = a$.

(III) There exists some $i_0 \in \mathcal{G}$ and an instance of (Pi)-(Piv) such that $p(p(i_0))$ exists. To show this case, let us initiate the tree construction method (Pi)-(Piv) with $p(p(i_0))$ as seed. If i_0 fulfills $N_{i_0 \setminus p(i_0)} = \emptyset$, then we take $v_0 \equiv i_0$. Otherwise, we take v_0 to be the first descendant of i_0 such that $N_{v_0 \setminus p(v_0)} = \emptyset$. If we use v_0 as the seed for a new tree construction process (Pi)-(Piv), we can see that this results in a method faster than that by Kirkley. In particular, the update process sums over neighborhoods that either are N_{v_0} or $N_{b \setminus p(b)}$ for some $b \in \mathcal{G} \setminus \{v_0\}$. However, since $N_{v_0} \subseteq N_{p(v_0) \setminus p(p(v_0))}$ (where we take $p(\cdot)$ in the sense of the first tree-equivalent method considered here), the update equations are at most as complex as those in the KCN-method. Regarding the inference equations, they are less complex since $N_{v_0} \subsetneq N_{p(v_0)}$ (this is the case since $p(p(v_0))$ exists) and $N_{b \setminus p(b)} \subsetneq N_b$.

(Tii) There exists an instance of the tree-equivalent method whose time complexity is larger than that of the KCN-method if and only $N_i^{\ell_0} \subsetneq \mathcal{G}$ for all $i \in \mathcal{G}$.

To show this, we begin considering necessity. To prove it, we argue by contrapositive. We assume that $N_i^{\ell_0} = \mathcal{G}$ for some $i \in \mathcal{G}$ and consider the following cases:

(I) If $N_i^{\ell_0} = \mathcal{G}$ for all $i \in \mathcal{G}$, then we can follow the proof of necessity for (Ti).

(II) There exists some $i_0 \in \mathcal{G}$ such that $N_{i_0}^{\ell_0} \subsetneq \mathcal{G}$. If this is the case, and we use as seed for the tree-equivalent method some $i_1 \in \mathcal{G}$ such that $N_{i_1}^{\ell_0} = \mathcal{G}$, then, following the proof of (I), the KCN-method offers no complexity advantage. If we fix some $i_0 \in \mathcal{G}$ such that $N_{i_0}^{\ell_0} \subsetneq \mathcal{G}$ and use it as seed for the tree-equivalent method, then the time complexity is twice that of summing over some neighborhood $N \subsetneq \mathcal{G}$ (one contribution coming from inference and one from updating). However, one can show that this complexity is bounded by that of the KCN-method, which has (at least) a contribution coming from summing over $N_{i_1 \setminus i_0}^{\ell_0}$ during the update and one from summing over $N_{i_1}^{\ell_0}$ during inference.

To conclude, we show sufficiency. In order to do so, fix $i_0 \in \mathcal{G}$ a vertex such that summing over $N_i^{\ell_0}$ is maximal for $i = i_0$, consider the tree-equivalent method with seed i_0 and note, moreover, that, by definition of i_0, and since $N_{i \setminus j}^{\ell_0} \subsetneq N_i^{\ell_0}$ for all $i \in \mathcal{G}$ and $j \in N_i^{\ell_0}$, the complexity of summing over $N_{i_0}^{\ell_0}$ is strictly larger than that of summing over $N_{i \setminus j}^{\ell_0}$ for all $i, j \in \mathcal{G}$. The complexity of the tree-equivalent method is twice that of summing over $N_{i_0}^{\ell_0}$, with one contribution coming from the update and the other from inference. However, although the complexity of inference in the KCN-method is exactly the same as that of this tree-equivalent instance, its update complexity is the maximum over $i \in \mathcal{G}$ and $j \in N_i^{\ell_0}$

of the one that sums over $N_{i\backslash j}^{\ell_0}$, which is strictly smaller than that of summing over $N_{i_0}^{\ell_0}$.

Note that the advantage in (Ti) works for most graphs of interest, since it only requires that there exists some vertex such that the graph not only consists of loops around this vertex. The advantage in (Tii) is similar, although it requires the stronger property that there is no vertex such that the graph consists of loops around this vertex.

Note that reducing the complexity of the KCN-method requires to first find a specific vertex $i_0 \in \mathcal{G}$ that we use as seed in the tree-equivalent method. While one could add this to the complexity, in practice (for instance in the context of decoding) it is not important since it can be effectively done and it only depends on the topology of the graph, that is, it can be found once and then used in the following as long as the topology remain unchanged.

Appendix B: Derivation of the inference equations for tensor networks

1. The largest loop length is bounded by the KCN-parameter

(Ei) follows since we have that

$$\begin{aligned} p_i(x) &= \frac{1}{Z_{i_1}} \operatorname{tr}_{\backslash x_i}\left(S_{i_1}\prod_{k\in S_{i_1}} m_{k\to i_1}\right) = \frac{1}{Z_{i_2}} \operatorname{tr}_{\backslash x_i}\left(S_{i_2}\prod_{k\in S_{i_2}} m_{k\to i_2}\right) \\ &= \frac{1}{2Z_{i_1}} \operatorname{tr}_{\backslash x_i}\left(S_{i_1}\prod_{k\in S_{i_1}} m_{k\to i_1}\right) + \frac{1}{2Z_{i_2}} \operatorname{tr}_{\backslash x_i}\left(S_{i_2}\prod_{k\in S_{i_2}} m_{k\to i_2}\right). \end{aligned} \tag{B1}$$

Although the first two equations are simpler, in the spirit of [14], we will give preference to the last equality (i.e. to Eq. (18)) in case the KCN-parameter ℓ_0 is strictly smaller than the largest loop length.

(Eii) is directly obtained via the following chain of equalities:

$$\begin{aligned} U &= \sum_x p(x)E(x) \\ &= -\sum_i \sum_x p(x)\log T_i(x_{\partial T_i}) \\ &= -\sum_i \sum_{x_{\partial T_i}} p(x_{\partial T_i})\log T_i(x_{\partial T_i}) \\ &= -\sum_i \frac{1}{Z_{\partial T_i}} \operatorname{tr}\left(\log\left(T_i\right) S_i \prod_{k\in S_i} m_{k\to i}\right). \end{aligned} \tag{B2}$$

(Eiii) can be shown in a few steps. First, before we argue why the equation holds, we first note that the denominator is well-defined, that is, that exchanging i and j in the denominator does not affect the computation. This is the case since $m_{k\to j} = m_{k\to i}$ for all $T_k \in S_{i\cap j} \setminus \{T_i, T_j\}$. (To see this, one has to notice that, if $T_k \in S_{i\cap j}$ and the KCN-parameter is larger than all loop lengths, then

$$S_{k\cap j} = S_{k\cap i} = S_{i\cap j}. \tag{B3}$$

Hence, $S_{k\backslash j} = S_k \setminus S_{k\cap i} = S_k \setminus S_{k\cap i} = S_{k\backslash i}$ and $m_{k\to j} = m_{k\to i}$.)

We show now the validity of (Eiii). If we define $n_{i\to j}$ such that $m_{i\to j} = n_{i\to j}/M_{i\to j}$ for all $i \in \mathcal{G}$ and $j \in S_i$, where $M_{i\to j}$ is the product of all the normalization constants that have been introduced into $m_{i\to j}$ during the iteration of the BP equations, then we have that

$$\begin{aligned} Z &= \operatorname{tr}\left(S_i \prod_{j\in S_i} n_{j\to i}\right) \text{ and} \\ Z &= \operatorname{tr}\left(S_{i\cap j} n_{j\to i} \prod_{k\in S_{i\cap j}} n_{k\to j}\right). \end{aligned} \tag{B4}$$

(Note that we have implicitly used Eq. (B4) when showing (Ei).)

Hence, disregarding for the moment the normalization constants, we have

$$\begin{aligned} &\frac{\prod_{i\in\mathcal{V}} \operatorname{tr}\left(S_i \prod_{j\in S_i} m_{j\to i}\right)}{\prod_{((i,j))\in\mathcal{G}} \operatorname{tr}\left(S_{i\cap j} m_{j\to i} \prod_{k\in S_{i\cap j}} m_{k\to j}\right)^{\frac{2}{|S_{i\cap j}|}}} \\ &\propto \frac{\prod_{i\in\mathcal{V}} Z}{\prod_{((i,j))\in G} Z^{\frac{2}{|S_{i\cap j}|}}} \\ &= \frac{Z^{|\mathcal{V}|}}{Z^{\sum_{\{((i,j))\in\mathcal{G}\}/\sim} \frac{2}{|S_{i\cap j}|} \sum_{t=1}^{|S_{i\cap j}|-1} t}} \\ &= \frac{Z^{|\mathcal{V}|}}{Z^{\sum_{\{((i,j))\in\mathcal{G}\}/\sim} |S_{i\cap j}|-1}} \\ &= \frac{Z^{|\mathcal{V}|}}{Z^{|\mathcal{V}|-1}} \\ &= Z, \end{aligned} \tag{B5}$$

where we use Eq. (B4) to go from line one to line two, Eq. (B3) and, for all pairs $((i,j)),((k,\ell)) \in \mathcal{G}$, the

equivalence relation $\sim$ with $((i,j)) \sim ((k,\ell))$ provided $S_{i\cap j} = S_{k\cap\ell}$ in the first equality, the arithmetic series sum identity in the second, and the fact that, if we enumerate the vertices in $\mathcal{G}$ via (Pi)-(Piv) using some seed $i_0 \in \mathcal{V}$, then each vertex $m \in \mathcal{V} \setminus \{i_0\}$ is only counted once in

$$\sum_{\{((i,j))\in\mathcal{G}\}/\sim} |S_{i\cap j}| - 1,$$

namely, for the equivalence class of $S_{m\cap p(m)}$, where $p(m)$ is the parent of m in the enumeration procedure. Hence, the sum equals $|\mathcal{V}| - 1$ and we get the third equality.

To conclude, we show that the normalization constants cancel out. In particular, we have that

$$\begin{aligned}
&\frac{\prod_{i\in\mathcal{V}}\prod_{j\in S_i} M_{i\to j}}{\prod_{((i,j))\in\mathcal{G}}\left(M_{j\to i}\prod_{k\in S_{i\cap j}} M_{k\to j}\right)^{\frac{2}{|S_{i\cap j}|}}} \\
&= \frac{\prod_{i\in\mathcal{V}}\prod_{\{j\in\mathcal{G}|((i,j))\in\mathcal{G}\}/\sim} M_{i\to j}^{|S_{i\cap j}|-1}}{\prod_{i\in\mathcal{G}}\left(\prod_{\{((k,\ell))\in\mathcal{G}|i\in S_{k\cap\ell} \text{ and } k,\ell\neq i\}} M_{i\to k}^{\frac{2}{|S_{k\cap\ell}|}}\right)\left(\prod_{\{((k,\ell))\in\mathcal{G}|\ell=i\}} M_{i\to k}^{\frac{2}{|S_{i\cap k}|}}\right)} \\
&= \frac{\prod_{i\in\mathcal{V}}\prod_{\{j\in\mathcal{G}|((i,j))\in\mathcal{G}\}/\sim} M_{i\to j}^{|S_{i\cap j}|-1}}{\prod_{i\in\mathcal{V}}\prod_{\{j\in\mathcal{G}|((i,j))\in\mathcal{G}\}/\sim} M_{i\to j}^{|S_{i\cap j}|-1}} \\
&= 1,
\end{aligned} \tag{B6}$$

where the first two equalities essentially follow from the fact that, whenever $i \in S_{j\cap k}$, then $m_{i\to\ell}$ is the same for all $\ell \in S_{j\cap k} \setminus \{i\}$, and the grouping into equivalent classes that we introduced in Eq. (B5).

2. The largest loop length is not bounded by the KCN-parameter

In this case, we use the same inference equations except for (Eiii), which we have to modify slightly. First, let us note that, under the assumption that the KCN-parameter bounds the loop length, we have that

$$\operatorname{tr}\left(S_i\prod_{j\in S_i} m_{j\to i}\right) = \operatorname{tr}\left(T_i\prod_{j\in S_i/\sim} p_{j\to i}\right)\times \prod_{j\in S_i}\left(\frac{\operatorname{tr}\left(S_{i\cap j}m_{j\to i}\prod_{k\in S_{i\cap j}} m_{k\to j}\right)}{\operatorname{tr}\left(T_i m_{i\to j}p_{j\to i}\right)}\right)^{\frac{1}{|S_{i\cap j}|-1}}$$

where, taking an appropriate x,

$$p_{j\to i}(x) \equiv \operatorname{tr}\left(\frac{S_{i\cap j}}{T_i}m_{j\to i}\prod_{k\in S_{i\cap j}\setminus\{i\}} m_{k\to j}\right).$$

Lastly, provided the loop bound is not fulfilled, we can estimate the partition function via

$$\prod_{((i,j))\in\mathcal{G}}\operatorname{tr}\left(S_{i\cap j}m_{j\to i}\prod_{k\in S_{i\cap j}} m_{k\to j}\right)^{\frac{1}{\binom{|S_{i\cap j}|}{2}}}\times \operatorname{tr}\left(T_i\prod_{j\in S_i/\sim} p_{j\to i}\right)^{1-\sum_{j\in S_i}\frac{1}{|S_{i\cap j}|-1}-\sum_{j\in\mathcal{NN}(i)}W_{i,j}}\times \prod_{(i,j)\in\mathcal{G}}\operatorname{tr}\left(T_iT_jm_{j\to i}m_{i\to j}q_{j\to i}\right)^{W_{i,j}},$$

where, taking an appropriate x,

$$q_{j\to i}(x) \equiv \operatorname{tr}\left(\frac{S_{i\cap j}}{T_iT_j}\prod_{k\in S_{i\cap j}\setminus\{i\}} m_{k\to j}\right)$$

and, denoting by χ the indicator function,

$$W_{i,j} \equiv 1 - \sum_{((\ell,k))\in\mathcal{G}}\frac{1}{\binom{|S_{\ell\cap k}|}{2}}\chi_{\{(i,j)\in N_{\ell,k}\}}$$

in order to avoid overcounting.

Appendix C: Less complex tensor network algorithm when the largest loop length is not bounded by the KCN-parameter

Whenever the KCN-parameter ℓ_0 is smaller than the maximal loop length, the update equations may require

more computational resources than they would if that was not the case. This happens since we ought to be able to contract over $S_{i\cap j}$ and $S_{i\setminus j}$ instead of only over $S_{i\setminus j}$, which may be computationally more expensive in some instances. An instance where this happens in the lattice. For example, if we take $N_i^{K,4}$ as neighborhoods (that is, a 3×3 block around each node i), the the number of both functions and variables required for updating increases. In particular, updating $m_{i\to j}^{(t)}$ and $m_{i\cap j\to i}^{(t)}$ involve, respectively, (at most) 16 and 17 variables. Although this is a modest difference, it may be significant in other instances, and the purpose of this section is to show how one may find a less complex algorithm.

In the lattice case, and in some other instances, we may be able by inspection to profit form a certain regularity in the graph to avoid the extra computational cost. To do this, in the case of the lattice with neighborhoods $N_i^{K,4}$, we incorporate to our BP scheme some messages that are sent from some actual nodes to some virtual nodes that we incorporate in the edges of the graph. To see why this is sufficient, it is enough to note that, if we denote each node by its $(x,|y|)$ coordinates with $(0,0)$ being the top left corner, then each node receives messages from its neighboring nodes along the edges that connects it to them according to Figure 9 (a).

The construction works as follows: We turn a $t\times t$ lattice $\mathcal{G}$ into a $(t+2)\times(t+2)$ **extended** lattice $\mathcal{G}'$ by adding the required virtual vertices and edges. The new messages are sent from non-virtual to virtual nodes treating $\mathcal{G}'$ as if it was the actual graph and using the neighborhoods accordingly. Aside from the messages already in the original lattice $\mathcal{G}$, we need to add the messages $m_{i\to j}$ with

$$(i,j)=\begin{cases}((x,1),(x,0)) & \text{for } 1\le x\le t,\\ ((x,t-1),(x,t)) & \text{for } 1\le x\le t,\\ ((1,y),(0,y)) & \text{for } 1\le |y|\le t, \text{ and}\\ ((t-1,y),(t,y)) & \text{for } 1\le |y|\le t.\end{cases}$$

An illustration of the resulting graph $\mathcal{G}'$ as well as of the newly incorporated messages can be found in Figure 9 (b).

Using the newly incorporated messages and profiting from the symmetry, we can avoid computing the intersection messages. We show how to do so in Figure 10.

It should be noted that, since we can run all message updates in parallel, the addition of new messages (which are as complex as the original ones) does not affect the complexity. Moreover, once the messages have converged or we run out of iterations, we can use the same inference equations as before, since they do not suffer from the missing legs issue.

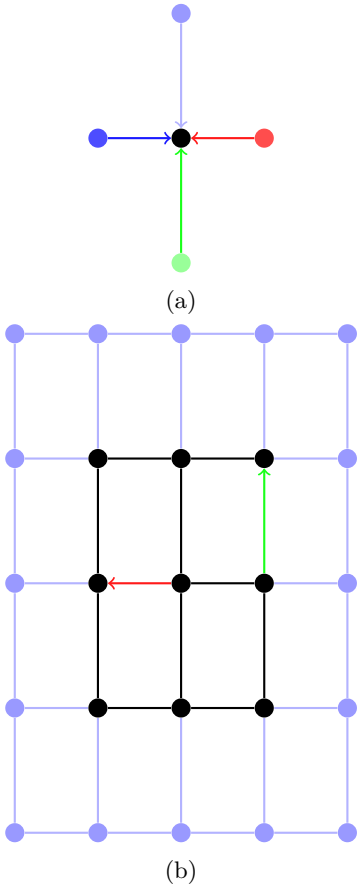


FIG. 9: (a) Messages sent to a node from its nearest neighbors. (b) Extended lattice $\mathcal{G}'$ formed by the original lattice $\mathcal{G}$ (in black) and the virtual nodes and edges (in blue).

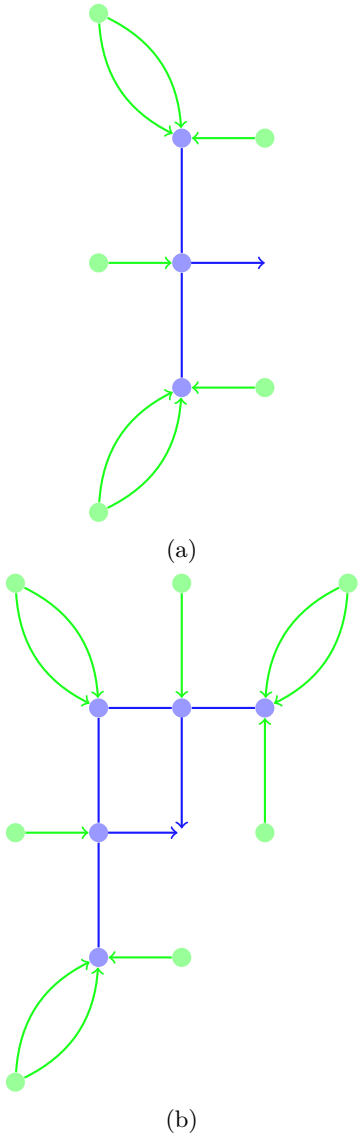
Appendix D: Mapping arbitrary graphical models to networks and tensor networks

1. General results

We consider here the association of an equivalent simpler graphical model to another one. In particular, we consider the two opposing sides of the spectrum: networks, where factors depend at most on two variables, and tensor networks, where variables are shared by, at most, two functions.

Although it is known not to be always optimal [37], it is customary to introduce algorithms for graphical models in the restricted case of pairwise potentials [9–12, 38]. In fact, it is known that for any graphical model there is an equivalent graphical model with pairwise potentials (see [39, Appendix A] or [38, theorem 4]). Moreover, we can extend the equivalence to graphical models with pairwise potentials that depend (at most) on three variables.

The reduction from graphical models to tensors networks has received less attention, with the closest to what we are interested in being [40]. As we will see, tensor net-



(a)

(b)

FIG. 10: Less complex update rules (cf. Figures 5 and 6) for a lattice $\mathcal{G}$ using the messages from its extended lattice $\mathcal{G}'$.

works play a role dual to that of networks, that is, any graphical model has an equivalent tensor networks whose tensors depend, at most, on three variables.

Given a graphical model

$$P(x_1,\dots,x_n)=\frac{1}{Z}\prod_{a\in\mathcal{F}} f_a(x_{\partial a}),$$

the following statements hold:

(Si) There exists an equivalent network $P_0 = P$. This was first noted in [39, Appendix A]. In particular, without loss of generality, their strategy can be visualized in [39, Figure 6].

(Sii) There exists an equivalent network $P_0 = P$ whose variables are shared, at most, by three factors. This can be found in [38] and the references therein. In particular, without loss of generality, their strategy can be visualized in [38, Equation 3.8].)

(Siii) In general, there is no equivalent network P_0 whose variables are shared, at most, by two factors. This was shown in [41]. (See also [38, Section 3.2.1].)

(Siv) There exists an equivalent tensor network $P_0 = P$. This can be achieved by introducing delta-tensors, as one can visualize in [40, Figure 3].

(Sv) There exists an equivalent tensor network $P_0 = P$ whose tensors depend, at most, on three variables. Since this was not stated before, we give a direct formal proof: By (Siv), we can associate to P a tensor network $P_1 = (f_b)_{b\in\mathcal{F}'}$. For each tensor f_b, we either create a copy $(f')^1_b = f_b$ provided $|\partial b| \le 3$ or, if $|\partial b| > 3$, we create a new set of variables $\{y^1_b,\dots,y^{|\partial b|-3}_b\}$, where $y^k_b \equiv (x^1_b,\dots,x^{k+1}_b)$ for $k = 1,\dots,|\partial b|-3$ and $x^1_b,\dots,x^{|\partial b|}_b$ are the variables connected to f_b. Then, for each y^k_b, we create a factor $(f')^k_b$ such that $(f')^1_b$ depends on x^1_b, x^2_b and y^1_b and is the product of two deltas

$$(f')^1_b(x^1_b,x^2_b,y^1_b)\equiv \delta_{x^1_b,(y^1_b)_1}\delta_{x^2_b,(y^1_b)_2}$$

then, for $1 < k \le |\partial b| - 3$, $(f')^k_b$ depends on x^{k+1}_b, y^{k-1}_b and y^k_b and is the product of $k+1$ deltas

$$(f')^k_b(x^{k+1}_b,y^{k-1}_b,y^k_b)\equiv \delta_{x^{k+1}_b,(y^k_b)_{k+1}}\prod_{j=1,\dots,k}\delta_{(y^{k-1}_b)_j,(y^k_b)_j},$$

and, lastly, the factor $(f')^{|\partial b|-3}_b$ depends on $x^{|\partial b|-1}_b, x^{|\partial b|}_b$ and $y^{|\partial b|-3}_b$ and is a copy of f_b

$$(f')^{|\partial b|-2}_b(x^{|\partial b|-1}_b,x^{|\partial b|}_b,y^{|\partial b|-3}_b)\equiv f_b(x^1_b,\dots,x^{|\partial b|}_b).$$

We conclude by noticing that

$$P_0\equiv((f')^k_b)_{b\in\mathcal{F}',k\in\{1,\dots,|\partial b|-2\}}$$

is a tensor network whose tensors have at most three legs and is equivalent to P. The transformation we have followed in this point can be visualized in Figure 11.

(Svi) In general, there is no equivalent tensor network P_0 whose tensors depend, at most, on two variables. Such a tensor network is equivalent to a network whose variables are shared, at most, by two factors, that is, the statement was shown in [41].

(Svii) A (connected) graphical model is a network and a tensor network if and only if its associated factor graph is a chain. (Recall that a **chain** is a graph which consists of an alternating sequence of vertices and edges $x_0, e_1, x_1, \dots, e_n, x_n$ starting and ending in vertices such that each edge is incident to the vertex immediately preceding and following it. See Figure 12.)

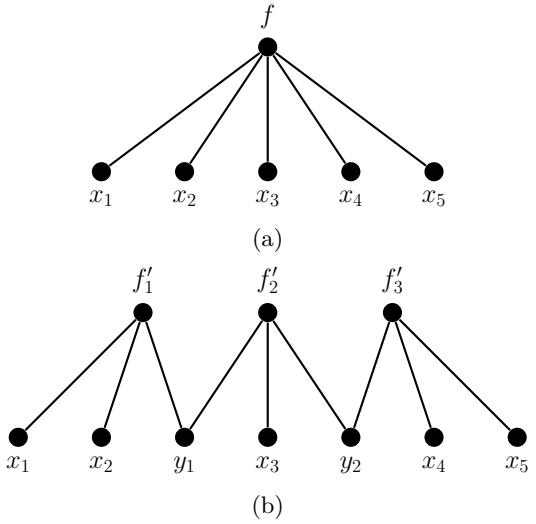


FIG. 11: A tensor network consisting of a tensor that depends on five variables (a) and an equivalent tensor network with tensors of degree at most three (b). The map from (a) to (b) is specified in (Sv).

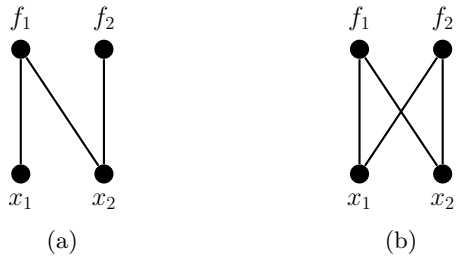


FIG. 12: A (connected) graphical model that is both a tensor network and a network is either an open (a) or a closed (b) chain.

Note that, alternatively, one can use (Sii) together with the idea behind (Si) to show (Sv). In particular, [38, Theorem 5] shows that any graphical model P has an equivalent graphical model P_1 with factors of degree at most two and variables of degree at most three. We can follow the idea behind (Si) independently for each variable of degree three to obtain a graphical model P_0 with factors of degree at most three and variables of degree at most two. In particular, we substitute each variable x connected to three functions f_1, f_2, f_3 by three copies of x, y_1, y_2, y_3, we connect connect f_i to y_i for $i = 1, \ldots, 3$ and introduce a delta tensor connected to y_1, y_2 and y_3.

Lastly, one can use (Sv) to show (Sii). In particular, by (Sv), there exists a graphical model P_1 equivalent to P whose variables have degree at most two and whose factor have degree at most three. In order to do so, we can independently turn each factor of degree three $f(x_1, x_2, x_3)$ into a set of factors of degree two, at the expense of introducing one variable of degree three $y_{1,2} \equiv (x_1, x_2)$. In particular, we can substitute f by the degree two factors

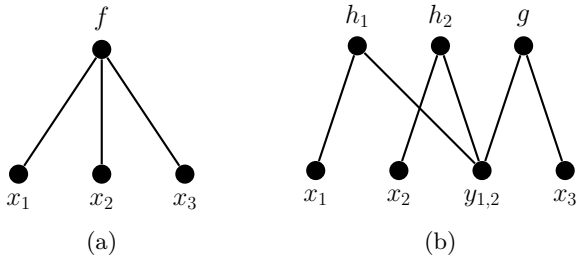


FIG. 13: A tensor network consisting of a tensor that depends on three variables (a) and an equivalent network with variables that are shared at most by tree factors (b). The map from (a) to (b) is specified at the end of Section D 1.

$g(x_3, y_{1,2}), h_1(x_1, y_{1,2})$ and $h_2(x_2, y_{1,2})$, where

$$
\begin{aligned}
g_1(x_3, y_{1,2}) &\equiv f((y_{1,2})_1, (y_{1,2})_2, x_3), \\
h_1(x_1, y_{1,2}) &\equiv \delta_{x_1,(y_{1,2})_1}, \\
h_2(x_2, y_{1,2}) &\equiv \delta_{x_2,(y_{1,2})_2}.
\end{aligned}
$$

We conclude by noting that the variables $y_{1,2}$ has degree three. The transformation we have followed in this point can be visualized in Figure 13.

2. Mapping a tensor network into a network and using the approach by Kirkley et al. directly

Turning a tensor network into a network and then using the KCN-approach is not optimal in terms of complexity. In particular, translating the tensor network into a network via the usual mappings (see the references within Appendix D 1) can increase the size of the loops and, hence, it results in larger neighborhoods over which we ought to compute correlations exactly in both the update and inference equations.

An instance of this issue, when the KCN-parameter bounds the loop length, can be found in in Figure 14, where both the space and time complexity increase when passing from a tensor network to a network. Note that, since both methods are exact in this case, we can compare them solely in terms of complexity.

Another instance of this issue, now with the KCN-parameter not bounding the loop length, can be found in in Figure 15, where both the space and time complexity increase again when passing from a tensor network to a network. It should be noted that, while neither of both methods is exact, the mapping into a network does not change the graph structure in any meaningful way and, hence, we do not expect BP to work better on the network than on the tensor network and we can again compare them solely in terms of complexity.

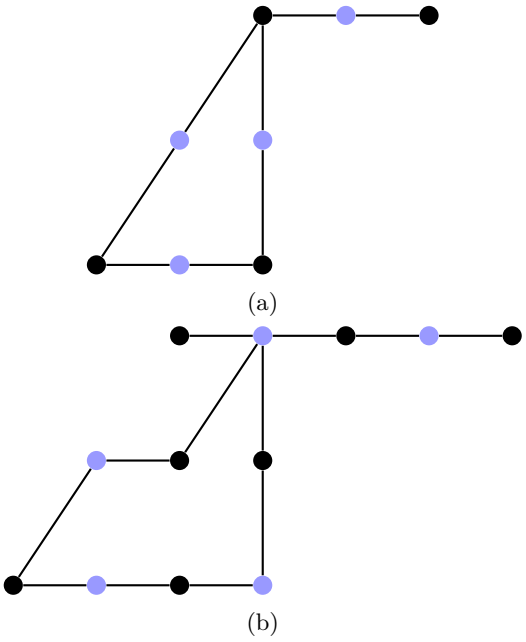


FIG. 14: A tensor network (a) and its associated network (b). The space and time complexity of updating (and similarly for inference) increase from $|X|^2$ and $|X|^3$ in (a) to $|X|^3$ and $|X|^6$ in (b). In this representation of graphical models [1], variables are represented by blue dots and factors by black ones, with the edges showing the connections among them.

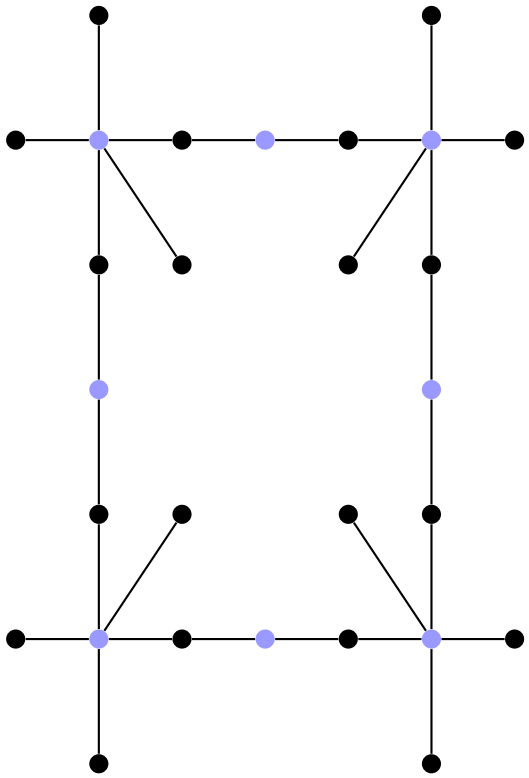


FIG. 15: The network associated to a square of a lattice tensor network. The space and time complexity of updating (and similarly for inference) increase from $|X|^2$ and $|X|^{17}$ in the original tensor network to $|X|^4$ and $|X|^{40}$ in the associated network.

Appendix E: The relation between the Kirkley et al. and the Wang et al. approaches

1. Neighborhoods

In this section, we consider the relation between the KCN- and WZPZ-neighborhoods. For simplicity, we will omit the variable that is used as a starting point when constructing the WZPZ-neighborhoods in our notation, that is, we will write N_ℓ^K (N_ℓ^W) instead of $N_i^{K,\ell}$ ($N_i^{W,\ell}$). Moreover, unless specified otherwise, all KCN- and WZPZ-neighborhoods use the same starting vertex.

The following statements hold:

(Wi) $N_\ell^K \subseteq N_{\ell-1}^W$ for all $\ell \geq 0$, where we take $N_{-1}^W \equiv N_0^W$. Moreover, in general, $N_\ell^K \not\subseteq N_{\ell-2}^W$.

To show the first statement, let us first note that, for $\ell < 3$, N_ℓ^K and $N_{\ell-1}^W$ consist of the point through which they are defined plus its nearest neighbors and the edges that connect it to them. Since the relation holds directly for those, we can assume $\ell \geq 3$. To show these cases, take some vertex $x \in N_\ell^K$ (we can argue analogously for edges). If x_0 is the starting points for the generation of the neighborhoods and x is a nearest neighbor of x_0, then $x \in N_{\ell_0}^W$ by definition. Hence, we can assume that x belongs to a path p_x of length smaller or equal to $\ell - 2$ between two nearest neighbors of x_0. Since the nearest neighbors of x_0 belong to $N_{\ell-1}^W$, provided $x \notin N_{\ell-1}^W$, then there would be two last vertices in p_x, y and z, such that $y, z \in N_{\ell-1}^W$, where by **last** we mean they are separated by more p_x edges from x_0. Given that they are connected to edges that do not belong to $N_{\ell-1}^W$, we have that $y, z \in \partial N_{\ell-1}^W$ and, in particular,

$$\min_{(a,b)} d_{a,b}(\partial N_{\ell-1}^W) \leq d_{y,z}(\partial N_{\ell-1}^W) \leq \ell - 2.$$

Hence, $x \in p_x \subseteq N_{\ell-1}^W$.

To show the second statement, it suffices to take $\ell = 3$ and a fully connected graph with three vertices.

(Wii) There exist graphs such that, respectively,

$$\{N_\ell^W\}_\ell \setminus \{N_\ell^K\}_\ell \neq \emptyset \text{ and } \{N_\ell^K\}_\ell \setminus \{N_\ell^W\}_\ell \neq \emptyset.$$

To see this, note that, given a 4×4 (or larger) lattice, $N_6^K \notin \{N_\ell^W\}_\ell$ (see Figures 16 and 17). For the opposite case, Figure 18 shows that $N_4^W(i_0) \notin \{N_\ell^K\}_\ell$, where i_0 is the initial point of the subset.

(Wiii) If there are no loops whose length is larger than ℓ_0, then $N_k^W \subseteq N_{\ell_0}^K$ for all $k \geq 0$.

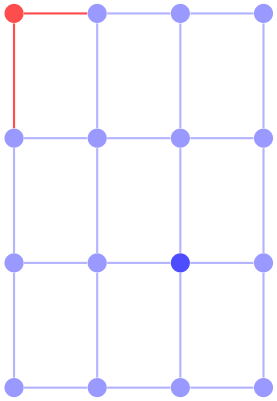


FIG. 16: The KCN-neighborhood with $\ell_0 = 6$ and initial node x_0 (dark blue) in a 4×4 lattice. The edges and nodes not in the neighborhood are represented in red.

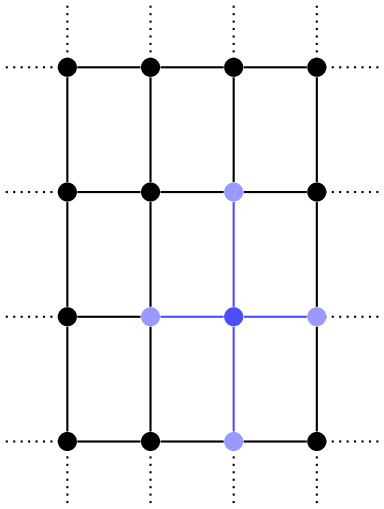
To show this statement, we will follow a constructive argument. Start by fixing some $k \geq 0$ and some vertex x_0 through which N_k^W is defined, and take some vertex $z \in N_k^W$ (the case of edges follows analogously). If z is a nearest neighbor of x_0, then $z \in N_{\ell_0}^K$ by definition. Otherwise, we construct a sequence of pairs of points $((z_1^t, z_2^t))_{t=1}^T$ and a sequence of paths $((p_1^t, p_2^t))_{t=1}^T$ as follows:

(I) We take z_1^1 and z_2^2 to be two points through which z is incorporated to N_k^W in its recursive definition (Ri)-(Rii) and take as p_1^1 (p_2^1) the path $z_1^1 - z$ ($z_2^1 - z$) through which z is incorporated to N_k^W.

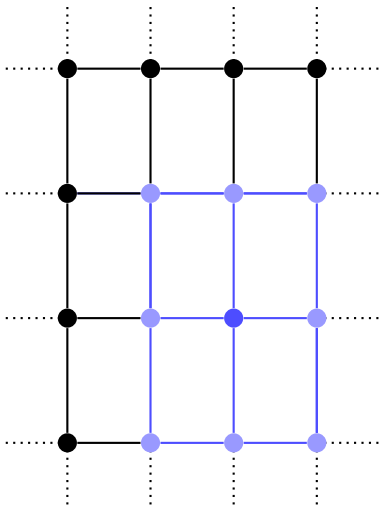
(II) For $0 \leq t$, we define z_1^{t+1} and z_2^{t+1} according to the following two cases:

(II.1) If $g(z_1^t) < g(z_2^t)$ (and analogously in the opposite case) and z_1^t and z_2^t are not both nearest neighbors of x_0, then we take $z_1^{t+1} = z_1^t$ and $p_1^t = \emptyset$. Moreover, if $a \neq z_1^{t+1}$ is one of the two points through which z_2^t is incorporated into N_k^W, then $z_2^{t+1} = a$ and we take as p_2^{t+1} the path $a - z_2^t$ through which z_2^t is incorporated to N_k^W.

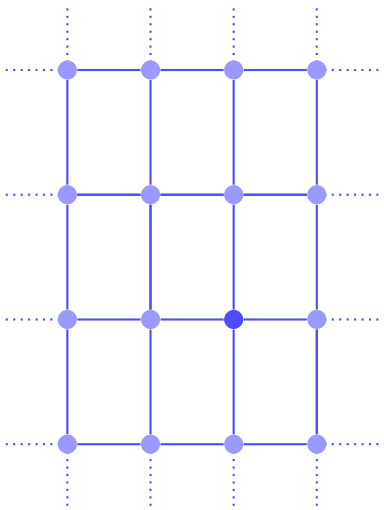
(II.2) If $g(z_1^t) = g(z_2^t)$, z_1^t and z_2^t are not both nearest neighbors of x_0, and a and b (c and d) are vertices through which z_1^t (z_2^t) are defined, then: If the paths used in the recursive definition of N_k^W to incorporate z_1^t do not intersect with those used to incorporate z_2^t, we can take $z_1^{t+1} = a$, $z_2^{t+1} = c$, p_1^{t+1} to be the path $a - z_1^t$ and p_2^{t+1} to be the path $a - z_2^t$. Otherwise, assuming the first intersection of the path $z_2^t - c$ is with the path $z_1^t - b$, we take $z_1^{t+1} = a$, $z_2^{t+1} = b$, p_1^{t+1} to be the path $a - z_1^t$ and p_2^{t+1} to be the path $b - i - z_2^t$, where i denotes the first edge where $z_2^t - c$



(a) $N_0(i_0)^W = N_1^W(i_0) = N_2^W(i_0)$.

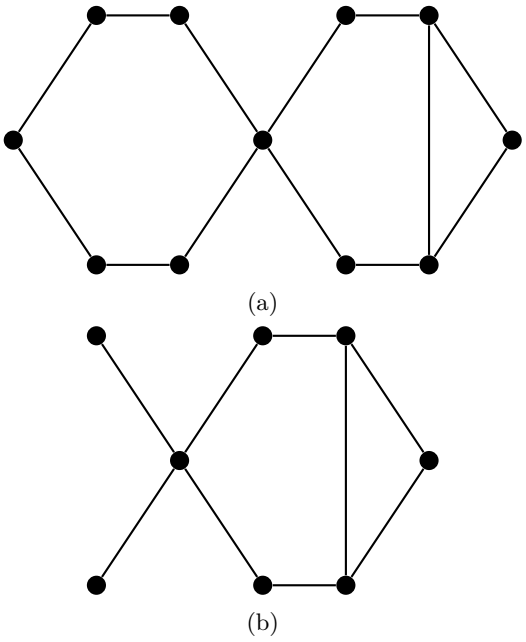


(b) $N_3^W(i_0)$.



(c) $N_k^W(i_0)$ for $\ell \geq 4$.

FIG. 17: The WZPZ-neighborhoods (in blue) centered around $i_0 \in \mathcal{G}$ (in dark blue) on a lattice $\mathcal{G}$ for different values of ℓ. For $\ell \geq 4$ and any central node and lattice size, $N_\ell^W = N_4^W = \mathcal{G}$.



(a) (b)

FIG. 18: A graph $\mathcal{G}$ (a) that has a WZPZ-neighborhood around the central node i_0, $N_4^W(i_0)$ (b), that does not belong to the family of KCN-neighborhoods.

and $z_1^t - b$ intersect.

(II.3) If z_1^t and z_2^t are not both nearest neighbors of x_0, then we take $T \equiv t$ and define p^0 to be x_0 together with the edges that join it to z_1^t and z_2^t, respectively.

We conclude noting that T is finite since $m_t \equiv \max\{g(z_1^t), g(z_2^t)\}$ is monotonically decreasing in t. Moreover, note that, w.l.o.g., we can assume that, for all $1 \leq t \leq T$, all edges and points in $p_1^t \setminus \{z_1^t\}$ ($p_2^t \setminus \{z_2^t\}$) have generation $g(z_1^{t-1})(g(z_2^{t-1}))$, where we take $z_1^0, z_2^0 \equiv z$. Because of this,

$$\widehat{p} \equiv p^0 \bigcup_{t=1,\ldots,T} \bigcup_{i=1,2} p_i^t$$

is a cycle such that $z, x_0 \in \widehat{p}$. In particular, by definition, $z \in N_{\ell_0}^K$.

(Wiv) If the loop length is bounded by ℓ_0, then, for all $y, z \in \partial N_{\ell_0-1}^W$, we have that $d_{y,z}(\partial N_{\ell_0-1}^W) = \infty$.

We will show the statement by contradiction. Under the hypothesis, and by properties (i) and (iii), we have that $N_{\ell_0-1}^W \subseteq N_{\ell_0}^K \subseteq N_{\ell_0-1}^W$. Hence,

$$N_{\ell_0}^K = N_{\ell_0-1}^W.$$

In case $d_{y,z}(\partial N_{\ell_0-1}^W) < \infty$, then, arguing as in (iii), we could construct a cycle around x_0 with vertices and edges outside $N_{\ell_0-1}^W = N_{\ell_0}^K$. This contradicts the definition of $N_{\ell_0}^K$.

(Wv) $N_\ell^W \subseteq N_t^K$ for all $\ell \geq 0$ provided

$$t \geq t_\ell \equiv \sum_{s=1}^{\ell-1} q_s s, \tag{E1}$$

where we take $q_1 \equiv 2$ and, for $s = 1, \ldots, \ell - 1$, q_s is the number of iterations that use s as length in the recursive construction of N_ℓ^W. Moreover, t_ℓ cannot be reduced.

To show the first statement, take a vertex $z \in N_\ell^W$ (note that an analogous argument can be done for edges). We can construct a loop that contains z and the point through which we define N_ℓ^W, x_0, following the algorithm in (Wiii). However, to ensure the bound Eq. (E1) is fulfilled, we need to slightly modify the algorithm to define $((z_1^t, z_2^t))_{t=1}^T$ taking into account the lengths of the paths involved in their definition. In particular, if $2 \leq s \leq \ell - 1$ is the length used in the generation of z_1^t and z_2^t, and the paths in the generation of z_1^t do not intersect those in the generation of z_2^t, we can pick as p_1^{t+1} and p_2^{t+1} the shortest paths in the generation of z_1^t and z_2^t, respectively, which then have a length of (at most) $s/2$. If the generation paths, which we denote by $p(z_1^t)$, $p'(z_1^t)$, $p(z_2^t)$ and $p'(z_2^t)$, intersect, then we assume w.l.o.g. that $p(z_1^t)$ intersects first with the generation paths for z_2^t and that its first intersection with them involves $p(z_2^t)$. We call the intersection vertex i and the distance from i to z_1^t along $p(z_1^t)$ ($p(z_2^t)$) d_1 (d_2). If $d_2 \leq d_1$ (otherwise, we do the converse), then we take p_2^{t+1} to be $p(z_2^t)$ between z_2^t and i and $p(z_1^t)$ from i onwards, and p_1^{t+1} to be $p'(z_1^t)$. By construction, p_2^{t+1} and p_1^{t+1} do not intersect each other and, if we denote the length of $p(z_1^t)$ by r, their combined length is, at most, $(s - r) + (d_2 + r - d_1) = s + d_2 - d_1 \leq s$. Lastly, note that, although we do not need to consider the iterations with $s = 1$ since they do not add to the loop length, we do take $q_1 = 2$ since the iterations stop at the nearest neighbors of x_0. In summary, $z \in N_t^K$ for all $t \geq t_\ell$.

To show the second statement, we provide in Figure 19 a graph in which the bound is attained for the case $\ell = 6$.

(Wvi) If there exists some ℓ_0 such that, if $y, z \in \partial N_{\ell_0}^W$, then $d_{y,z}(\partial N_{\ell_0}^W) = \infty$, then the loop length is bounded by t_{ℓ_0}.

To show this, note that, by hypothesis, if there is a loop around x_0, the point we use to initialize $N_{\ell_0}^W$, then the loop must use edges and vertices within $N_{\ell_0}^W$. As argued when showing property (Wv), the length of such loops is bounded by t_{ℓ_0}.

(Wvii) If there exists some ℓ_0 such that, if $y, z \in \partial N_{\ell_0}^W$, then $d_{y,z}(\partial N_{\ell_0}^W) = \infty$, then the inference formulas derived for $N_{t_{\ell_0}}^K$ are also exact when using

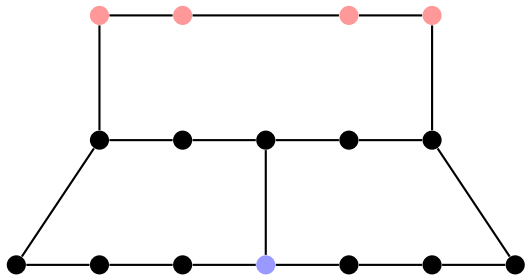


FIG. 19: A graph where the bound in Eq. (E1) is attained for $\ell = 6$. If we take as starting point x_0 the central bottom node (in blue), then any of the four z in the upper arc (in red) are included in N_{12}^K but not in N_t^K for $t < 12$.

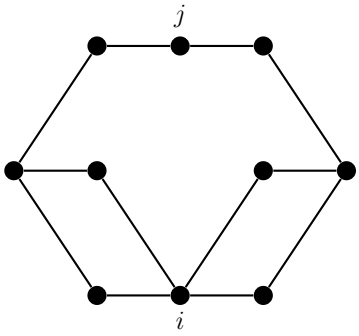


FIG. 20: A graph $\mathcal{G}$ where, taking $\ell_0 = 5$, $j \in N_i^{W,\ell_0} = \mathcal{G}$ although $i \notin N_j^{W,\ell_0}$. In particular, N_j^{W,ℓ_0} only includes j, its nearest neighbors and the edges connecting j to them.

$N_{\ell_0}^W$. This is the case since, as argued before, we have that $N_{\ell_0}^W = N_{t_{\ell_0}}^K$. Since everything is defined through these sets, the inference equations are exact and the same as in the case of KCN-neighborhoods.

As a last point, we note that WZPZ-neighborhoods are not as useful as KCN-neighborhoods when exchanging complexity for precision. This is the case since, althoguh $N_\ell^K \subseteq N_{\ell-1}^W$ for all $\ell \geq 0$ as showed in (Wi), we can see, for instance in the case of the lattice, that the WZPZ-neighborhood of any vertex is already the whole graph for $\ell \geq 4$. (See Figure 17.) This is the case since, once two vertices connected by (a horizontal, say) edge e are included, so are the two vertices along the immediate parallel edges e_1 and e_2. We can argue analogously for the vertical edges' case.

2. Inference equations

The inference equations from the KCN methdod can be applied directly except for the fact that, sometimes, the equations may need to be interpreted before using them, provided the bound on the loop length is not fulfilled. For instance, note that, if $\min_{(a,b)} d_{a,b}(\partial N_i^{W,\ell_0}) < \infty$, the WZPZ-neighborhoods do not in general have the property that $i \in N_j^{W,\ell_0}$ if and only if $j \in N_i^{W,\ell_0}$, as one can see in Figure 20. However, the equation for the partition function still works if we, for instance, take the pairs $((i,j)) \in \mathcal{G}$ to mean that either $i \in N_j$ or $j \in N_i$ and we modify the parameters $W_{i,j}$ accordingly.

Appendix F: Derivation of the partition function for the tree-equivalent approach

Let us take $\widehat{m}_{i\to j} = n_{i\to j}/\widehat{M}_{i\to j}$, where $\widehat{M}_{i\to j}$ is the product of all the normalization constants that have been introduced into $m_{i\to j}$ during the iteration of the BP equations and the rescaling added in order for the messages to fulfill Eq. (34).

If we assume that the blocks in BlockBP form a tree, then

$$Z = \operatorname{tr}\left(S_i \prod_{j\in\mathcal{NN}(i)} n_{j\to i}\right)$$

for all $i \in \mathcal{G}$. Moreover, by Eq. (34),

$$\widehat{M}_{i\to j}\widehat{M}_{j\to i} = \operatorname{Tr}\left(n_{i\to j}n_{j\to i}\right) = Z.$$

Putting these together we get that

$$\begin{aligned}
&\prod_{i\in\mathcal{G}} \operatorname{tr}\left(S_i \prod_{j\in\mathcal{NN}(i)} \widehat{m}_{j\to i}\right)\\
&= \prod_{i\in\mathcal{G}} \operatorname{tr}\left(S_i \prod_{j\in\mathcal{NN}(i)} n_{j\to i}\right) \prod_{j\in\mathcal{NN}(i)} \frac{1}{\widehat{M}_{j\to i}}\\
&= \left(\prod_{i\in\mathcal{G}} Z\right)\left(\prod_{i\in\mathcal{G}, j\in\mathcal{NN}(i)} \frac{1}{\widehat{M}_{j\to i}}\right)\\
&= Z^{|\mathcal{V}|}\frac{1}{Z^{|\mathcal{V}|-1}}\\
&= Z,
\end{aligned}$$

where, in the second to last equality, we have used the fact that a tree with $|\mathcal{V}|$ vertices has $|\mathcal{V}| - 1$ edges [42, Theorem 9.1].

If the graph after grouping is not a tree, then the same formula is used to approximate the partition function. In fact, instead of this simple derivation, it was shown that this formula corresponds to the Bethe approximation [27].

Appendix G: The Kirkley et al. method for a double layer complex-valued tensor network

A complex-valued tensor network of interest $T = (T_i)_i$ [19, 23, 25, 30] is the one that corresponds to the the scalar $||\psi||^2 = \langle\psi|\psi\rangle$ for some PEPS $|\psi\rangle$. This tensor network essentially corresponds to two copies of the same graph with edges joining each pair of identical nodes (cf. [30, Figure 1 (b)]). In order to follow the idea in our version of the Kirkley method for tensor networks and to have positive semi-definitive messages, we may define neighborhoods in a somewhat different way. In particular, given a tensor $T_i \in |\psi\rangle$, we define its neighborhood $N_{T_i \cup T_i^*}$ for some KCN-parameter ℓ_0 as the union of N_{T_i}, the ℓ_0-neighborhood we get if we apply the KCN-method centered at T_i to $|\psi\rangle$, together with $N_{T_i^*}$, the ℓ_0-neighborhood we get if we apply the KCN-method centered at T_i^* to $\langle\psi|$, and the edges that connect N_{T_i} with $N_{T_i^*}$. In order to define messages, given some pair $T_j, T_j^* \in N_{T_i \cup T_i^*}$, we consider all the edges that are connected either to T_j or to T_j^* and that are not contained in $N_{T_i \cup T_i^*}$. With these changes in mind, one can naturally change our version of the KCN method for tensor networks and obtain a generalization of the approach to these sort of tensor networks in the literature [19, 23, 25, 30].

Appendix H: Tree-equivalent method with intersection sets

In case the loops are bounded by the KCN-parameter, one can construct another tree-equivalent method that requires less complexity and retains exactness. However, although one can use the idea in this section to reduce the complexity in the Kirkley method provided the bound is fulfilled, the method does not generalize nicely to the case where the bound is not fulfilled. Nonetheless, for completeness, we sketch it here in the case of networks (one can analogously use it for graphical models in general).

The basic idea profits from the equivalence classes introduced in Appendix B. In particular, the exhaustive sums one needs to perform are not those in $N_{i\setminus j}$ but those in $N_{i\cap k} \equiv N_i \cap N_k$ for $k \in N_{i\setminus j} \setminus \{i\}$. In fact, one only needs to consider the equivalence classes of the intersection neighborhoods under $\sim$ (see Appendix B). With this is mind, one can build a tree-equivalent method with the set of messages

$$\begin{aligned} &\{m_{\overline{i\cap j}\to i}\}_{i\in V, j\in N_i/\sim, t\geq 0}, \\ &m_{\overline{i\cap j}\to i} : X \to \mathbb{R}_{\geq 0} \end{aligned} \tag{H1}$$

uniformly initialized, and updated, for $t \geq 0$, and taking as $S_{\overline{i\cap j}}$ the set of functions in $N_{\overline{i\cap j}}$, according to the following equation:

$$m^{(t+1)}_{\overline{i\cap j}\to i}(x_i) \equiv \frac{1}{M^{(t+1)}_{\overline{i\cap j}\to i}} \operatorname{tr}_{\setminus x_i}\left(S_{\overline{i\cap j}} \prod_{k\in N_{\overline{i\cap j}\setminus\{i\}}} \prod_{\ell \not\sim j} m^{(t)}_{\overline{k\cap \ell}\to j} \right), \tag{H2}$$

where $M^{(t+1)}_{\overline{i\cap j}\to i}$ is a normalization constant.