



## OPEN ACCESS

## EDITED BY

Federico Ricci-Tersenghi,  
Sapienza University of Rome, Italy

## REVIEWED BY

Pan Zhang,  
Chinese Academy of Sciences (CAS), China  
Alfredo Braunstein,  
Polytechnic University of Turin, Italy

## \*CORRESPONDENCE

Stefanos Kourtis,  
✉ stefanos.kourtis@usherbrooke.ca

RECEIVED 13 May 2024

ACCEPTED 03 October 2024

PUBLISHED 25 October 2024

## CITATION

Lanthier B, Côté J and Kourtis S (2024) Tensor  
networks for  $p$ -spin models .  
*Front. Phys.* 12:1431810.  
doi: 10.3389/fphy.2024.1431810

## COPYRIGHT

© 2024 Lanthier, Côté and Kourtis. This is an  
open-access article distributed under the terms  
of the [Creative Commons Attribution License  
\(CC BY\)](#). The use, distribution or reproduction in  
other forums is permitted, provided the original  
author(s) and the copyright owner(s) are  
credited and that the original publication in this  
journal is cited, in accordance with accepted  
academic practice. No use, distribution or  
reproduction is permitted which does not  
comply with these terms.

# Tensor networks for $p$ -spin models

Benjamin Lanthier, Jeremy Côté and Stefanos Kourtis\*

Institut quantique & Département de physique, Université de Sherbrooke, Sherbrooke, QC, Canada

We introduce a tensor network algorithm for the solution of  $p$ -spin models. We show that bond compression through rank-revealing decompositions performed during the tensor network contraction resolves logical redundancies in the system exactly and is thus lossless, yet leads to qualitative changes in runtime scaling in different regimes of the model. First, we find that bond compression emulates the so-called leaf-removal algorithm, solving the problem efficiently in the “easy” phase. Past a dynamical phase transition, we observe superpolynomial runtimes, reflecting the appearance of a core component. We then develop a graphical method to study the scaling of contraction for a minimal ensemble of core-only instances. We find subexponential scaling, improving on the exponential scaling that occurs without compression. Our results suggest that our tensor network algorithm subsumes the classical leaf removal algorithm and simplifies redundancies in the  $p$ -spin model through lossless compression, all without explicit knowledge of the problem’s structure.

## KEYWORDS

spin glass (theory), tensor network algorithms, disordered magnetic systems, satisfiability (SAT), model counting

## 1 Introduction

Spin glass physics appears in disciplines far-removed from its origin in condensed matter, including theoretical computer science [1], biology [2], and machine learning [3]. Spin glass models are generally easy to describe, yet hard to solve. One reason is that such models exhibit rugged energy landscapes [4], trapping optimization algorithms in local minima and leading to exponentially long run times.

A notable counterexample is the  $p$ -spin model [5], which is in fact easy to solve [6]. By mapping the model to a linear system of equations modulo 2, Gaussian elimination (GE) allows one to obtain the zero-temperature partition function of the model in polynomial time. While this model is a restricted version of a general spin glass model, its tractable analysis provides useful insights into the physics of spin glasses. Yet the  $p$ -spin model also exhibits rugged energy landscapes in certain regimes of the parameters, which is why it is a standard benchmark for classical [7–9] and quantum [10–15] optimization algorithms. In these regimes, simulated annealing fails or is inefficient for any  $p > 2$  [13, 15], and the same is true for quantum annealing [11], even when no phase transition is encountered [15]. Boolean satisfiability and local solvers also struggle with these models [16–21].

In this work, we introduce a tensor network algorithm for solving  $p$ -spin models. A tensor network (TN) is a data structure that allows for compact representation of a given (weighted) graphical model, including (quantum) spin Hamiltonians and constraint satisfaction problems, and whose contraction amounts to a (weighted) count of the solutions to the model [22–26]. While exact TN contraction is computationally hard in general even for restricted graph classes, such as planar grids [27], techniques involving tensor compression can lead to accurate and efficient approximate estimation of classical

partition functions or quantum expectations in specific cases [28–30]. TN methods have also previously been used in mean-field studies of graphical models and disordered spin Hamiltonians [31, 32].

Here, we show that compressed TN contraction applied to the  $p$ -spin model automatically emulates previously discovered algorithms for the solution of the model in its different phases. In contrast to previous works, the compression we perform is exact, meaning that it only resolves and simplifies redundancies in the TN at each step without loss of information. We illustrate the above with an application to the 3-spin model, in which the average number of interactions per spin  $\alpha$  controls transitions to different thermodynamic phases in the structure of the problem [5]. We find that compressed TN contraction automatically implements the leaf removal algorithm [5] and thus efficiently solves the problem when  $\alpha < \alpha_d$ , at which point a dynamical transition occurs. In contrast, compressed TN contraction scales superpolynomially when  $\alpha > \alpha_d$  but improves substantially on the exponential scaling of TN contraction without compression. We further show that when  $\alpha \in [2/3, 3/4]$ , compressed TN contraction outperforms naive GE. Finally, by devising a graphical scheme that exactly captures the dynamics of compressed TN contraction in the special case of spins appearing in exactly two interaction terms, for which no leaf removal occurs, we show numerically that the TN method solves the problem in subexponential time.

## 2 Definitions

### 2.1 The $p$ -spin model

We can write the  $p$ -spin model by specifying a bipartite graph  $G = (U, V, E)$ , where  $U$  is the set of nodes representing the  $n = |U|$  spins,  $V$  is the set of nodes representing the  $m = |V|$  interaction terms, and  $E$  is the set of edges connecting spin nodes to interaction nodes. We can then write the Hamiltonian of the  $p$ -spin model as:

$$H = \frac{1}{2} \left[ m - \sum_{v \in V} J_v \prod_{u \in N(v)} \sigma_u \right], \quad (1)$$

where  $J_v \in \{-1, 1\}$  are the couplings for the interaction at node  $v$ ,  $\sigma_u \in \{-1, 1\}$  is the value of the spin at node  $u$ , and  $N(v)$  is the set of  $p$  neighbours for the interaction described by node  $v$ . The minimum energy is zero, and it occurs when every interaction satisfies  $J_v = \prod_{u \in N(v)} \sigma_u$ . In this paper, we are interested in counting the number of zero-energy configurations for a given ensemble of bipartite graphs, that is, evaluating the zero-temperature partition function of the model.

By letting  $\sigma_u = (-1)^{x_u}$  and  $J_v = (-1)^{b_v}$ , we can rewrite the search for zero-energy configurations from Equation 1 as

$$A\vec{x} = \vec{b} \pmod{2}, \quad (2)$$

where  $A \in \{0, 1\}^{m \times n}$  is the biadjacency matrix of the graph  $G$ , with  $A_{vu} = 1$  indicating  $u \in N(v)$  and zero otherwise,  $\vec{x} \in \{0, 1\}^n$  encodes the spin configuration, and  $\vec{b} \in \{0, 1\}^m$  encodes the couplings. Finding the zero-energy configurations for Equation 1 is equivalent to solving the matrix Equation 2. Counting the number of configurations also involves manipulating Equation 2.

With this form, we can then cast the problem into the language of Boolean satisfiability (SAT), which we detail below.

### 2.2 The $\#p$ -XORSAT problem

#### 2.2.1 Definition

In its most general form, a SAT problem is the problem of deciding whether a logic formula built from a set of boolean variables  $\{x\} = \{x_1, x_2, \dots, x_n\}$  and the operators  $\wedge$  (conjunction),  $\vee$  (disjunction), and  $\neg$  (negation) evaluates to true, i.e., is satisfiable [33]. The SAT problem is characterized by the conjunction of clauses, each comprising disjunctions of variables where the negation operator may be applied. The SAT problem is NP-complete, and the same is true for many of its variants.

The constraint stipulating that every clause must consist of exactly  $p$  variables defines the  $p$ -SAT problem, which is also NP-complete. Counting the number of solutions that satisfy a given SAT problem, if any exist, defines the #SAT problem, which is even more challenging, falling under the #P-complete class. This property extends to  $\#p$ -SAT problems for  $p \geq 2$ .

The variant of the  $\#p$ -SAT problem that lets us count the number of zero-energy configurations of a given  $p$ -spin model is the  $\#p$ -XORSAT problem, defined below.

The  $\#p$ -XORSAT problem requires only a modification of the operators within the clauses from the standard  $p$ -SAT formulation. The disjunction is replaced by the exclusive-or (XOR) operator, which is mathematically represented by the summation modulo 2 operator ( $\oplus$ ). Given  $A$  and  $\vec{b}$  as in Equation 2, we can define an instance  $\phi$  of the  $p$ -XORSAT problem as:

$$\begin{aligned} \phi(\{x\}) &= \prod_{i=1}^m c_i, \\ c_i &= 1 \oplus b_i \oplus A_i \cdot \vec{x}, \\ \vec{x} &= (x_1, x_2, \dots, x_n) \in \{0, 1\}^n, \end{aligned} \quad (3)$$

where  $A_i \in \{0, 1\}^n$  is the  $i$ -th row of  $A$  and  $b_i$  is the  $i$ -th component of  $\vec{b}$ ,  $A_i \cdot \vec{x}$  indicates the dot product between  $A_i$  and  $\vec{x}$  (modulo 2), and  $c_i = 1$  implies the clause is satisfied ( $b_i \oplus A_i \cdot \vec{x} = 0$ ).

When one generates  $A$  by placing  $p$  ones in each row uniformly at random with no repeated rows and uniformly chooses  $\vec{b} \in \{0, 1\}^m$ , the clause density  $\alpha \equiv m/n$  characterizes much of the problem. In particular,  $p$ -XORSAT has two phase transitions [5]. The first occurs at  $\alpha_d$ , which indicates a dynamical transition in the structure of the solution space by dividing solutions into well-separated (in Hamming distance) clusters. The second occurs at the critical transition  $\alpha_c$ , where, with high probability, any instance becomes unsatisfiable (no solutions). This point signifies a similar phase transition even when  $\vec{b} = 0$ , meaning the configuration  $\vec{x} = 0$  is always a solution [20]. For  $p = 3$ , the constants are  $\alpha_d \approx 0.818$  and  $\alpha_c \approx 0.918$  [5].

#### 2.2.2 Gaussian elimination

Given a  $p$ -XORSAT instance  $\phi(\{x\})$ , as defined in Equation 3, we first translate it into the form of Equation 2. Then, we apply GE on the augmented matrix  $[A|\vec{b}]$ . If the system is inconsistent, there are no solutions. Otherwise, the solution count is:

$$\#\text{Solutions} = 2^{n - \text{rank}(A)}, \quad (4)$$

where all operations are modulo 2, as in applying GE. # $p$ -XORSAT is thus in P since it can be solved using Equation 4 in at most  $\mathcal{O}(n^3)$  time and  $\mathcal{O}(n^2)$  memory.

In Ref. [34], the authors studied the time and memory requirements for solving Equation 2 for  $p = 3$  using a “simple” version of GE. This version solves the linear equations in the order they appear in Equation 2 and with respect to a random variable. The authors showed that this simple algorithm will solve the problem in  $\propto n$  time and memory when  $\alpha \leq 2/3$ , and in  $\propto n^3$  time and  $\propto n^2$  memory when  $\alpha > 2/3$ .

The authors also presented a “smart” version of GE, where one first looks for the variable appearing in the least number of equations left to be solved (ties broken arbitrarily), then solves for that variable and substitutes it into the remaining equations. They argued that this smarter version of GE will solve the problem in  $\propto n$  time and memory when  $\alpha < \alpha_d$ , and in  $\propto n^3$  time and  $\propto n^2$  memory when  $\alpha > \alpha_d$ .

When one solves an equation that contains a variable which only appears in that equation, one can interpret the process graphically as a “leaf removal” algorithm [5]. We describe it below because it provides intuition as to why the “smart” version of GE is more efficient and will help explain the behaviour of our TN algorithm.

### 2.2.3 Leaf removal

Suppose we have an instance for  $p = 3$  and the variable  $x$  only appears in the linear equation  $x \oplus y \oplus z = b$ . No matter what values  $y$  and  $z$  take, it is always possible to choose  $x$  to make the equation true. We can therefore solve this equation for  $x$ , and only fix it once we have solved the rest of the (fewer) linear equations. But removing this equation may now cause  $y$  or  $z$  to only appear in a single other equation, so we solve those equations for  $y$  and  $z$ , and then what remains is an even smaller linear system. The process will continue until the remaining variables participate in at least two equations. In terms of the matrix  $A$  in Equation 2, each column will have at least two 1s (Note that if a variable appears in no equations it is, in essence, not part of the problem and so we can ignore it and simply multiply the count by 2.)

This algorithm is called leaf removal [5], and it allows us to simplify the  $p$ -XORSAT problem. Graphically, the algorithm begins with the bipartite graph  $G$  representing the problem, then iteratively finds variable nodes  $u \in U$  such that  $\text{deg}(u) = 1$ , and deletes the clause node  $v \in N(u)$  and  $v$ 's associated edges. The algorithm continues until either no clause nodes remain (and therefore, no edges) or a “core” remains, a subgraph of  $G$  where each variable node has degree at least two. One can then construct a solution to the original formula by working backwards from a solution to the formula corresponding to the core graph.

In Ref. [5], the authors showed that, for the ensemble where  $p = 3$  and one picks each clause uniformly at random from the  $\binom{U}{3}$  distinct tuples of variables, the leaf algorithm will succeed in reducing the corresponding graph to the empty graph when  $\alpha < \alpha_d \approx 0.818$ . Because at each step of the algorithm one can fix a variable node of degree 1 in order to remove a clause node, when no core remains the count will be  $2^{n-m}$ , where  $m$  is the number of clauses (or variables we have fixed). When  $\alpha > \alpha_d$ , a core will remain, which means leaf removal is not enough to solve the entire problem. The value  $\alpha_d$  indicates a dynamical transition in the problem, and it corresponds to a change in the structure of the set of solutions. The

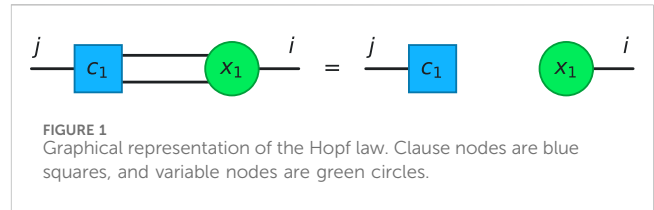


FIGURE 1 Graphical representation of the Hopf law. Clause nodes are blue squares, and variable nodes are green circles.

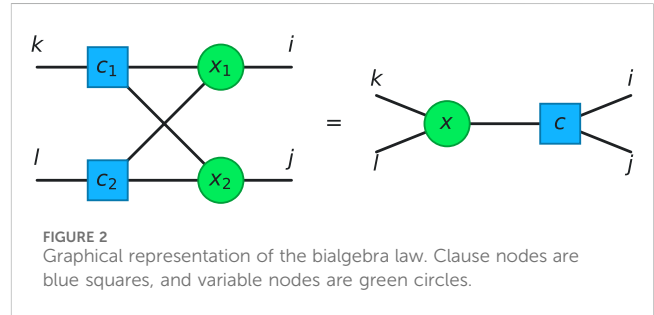


FIGURE 2 Graphical representation of the bialgebra law. Clause nodes are blue squares, and variable nodes are green circles.

“smart” GE uses this principle to achieve a speedup over the standard version.

We also note that when no core remains at the end of leaf removal, one can interpret the algorithm as finding a permutation of the rows and columns of the matrix  $A$  such that one can transform  $A$  into triangular form. Suppose the variable  $x_i$  only appears in equation  $j$ . One would then permute the rows 1 and  $j$  of  $A$ , as well as the columns 1 and  $i$ . Ignoring the first row and column of  $A$ , repeat the same procedure. Continuing in this way will yield a matrix  $A'$  which is in triangular form and has the same rank as  $A$ . The triangular form of  $A'$  implies that its rank is simply the number of rows, allowing one to calculate the number of solutions.

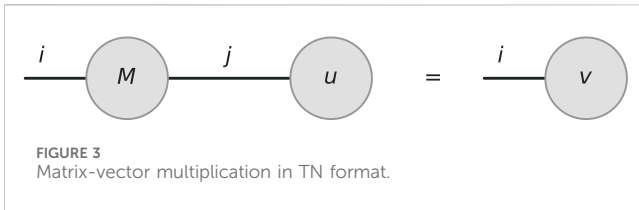
In the case of the  $p$ -XORSAT problem, this algorithm demonstrates that we can graphically identify and eliminate redundancy, reducing the problem’s size by focusing on the remaining core. Graphically, this problem does not only exhibit this rank-1 variable redundancy; two more are explained in the following section.

### 2.2.4 Graphical simplifications

There exist graphical rules, such as the leaf removal explained in Section 2.2.3, that let us simplify a  $p$ -XORSAT problem. These will be used in Section 3.3, where we develop a complementary graphical method for TN contraction. Note that we will study the case where  $\vec{b} = 0$  for simplicity. Then, we have the following examples of simplifications.

The first example is the Hopf law [35], where a clause involves the same variable multiple times. In this case, since  $i \oplus i = 0$  for boolean indices, when there are  $t$  occurrences of a variable in a clause, only  $t \bmod 2$  of them are necessary and the rest are redundant. In Figure 1, we show an example for  $t = 2$ .

The second example is the bialgebra law [35], where a set of clause nodes are all connected to a set of variable nodes. An example for two clauses and two variables is shown in Figure 2. These structures simplify to a single clause and single variable, as shown in Figure 2.



These simplifications correspond to eliminating redundancies in the problem. Resolving these redundancies can be exploited to solve the problem faster.

## 2.3 Tensor networks

TNs are a data structure that encodes a list of tensor multiplications. Intuitively, one can imagine a TN as a graph where each node represents a tensor, and edges represent the common axes along which one multiplies two tensors<sup>1</sup>. By contracting together neighboring nodes—multiplying the corresponding tensors together—one can sometimes efficiently compute a variety of quantities, making it a useful numerical method. Originally developed to efficiently evaluate quantum expectation values and partition functions of many-body systems, this tool now has applicability in many domains, including quantum circuit simulation [36] and machine learning [37]. As shown in [22], this tool can also be used for  $p$ -SAT problems.

For our work, contracting all of the tensors in the network together will yield the number of solutions to Equation 2. Below, we review the main ideas for TN methods that are relevant for us and determine the performance of our algorithm. These elements are: how to perform contractions, the importance of contraction ordering, and how to locally optimize the sizes of the tensors (which affect the memory requirements). We then describe our TN algorithm for the # $p$ -XORSAT problem in Section 3.1.

### 2.3.1 Contraction

A single tensor is a multidimensional array of values. Graphically, the number of axes (or rank) of the tensor is the degree of the corresponding node, and the size of the tensor is the number of elements (the product of the dimensions of the axes). The size of the TN is then the sum of all the tensor sizes. For any TN algorithm, one must keep track of the size of the TN to ensure the memory requirements do not exceed one's computational limits. In particular, one must consider how contracting tensors together changes the TN's size.

A simple example of contraction is the matrix-vector multiplication, which is represented graphically in Figure 3. Here, the vector  $\vec{u}$  (a rank-1 tensor) is represented by a node with a single edge connected to it and the matrix  $M$  (a rank-2 tensor) is also a

node, but with two edges. The matrix-vector multiplication shown in Figure 3 can also be written as the following summation:

$$\sum_j M_{ij} u_j = v_i. \quad (5)$$

In general, one can write the contraction of a TN by this summation over all the common (shared) axes. We will sometimes call tensors with common axes adjacent, in reference to a TN's graphical depiction.

When contracting tensors where each axis has the same dimension, we can graphically determine the resulting size by looking at the degree of the new node. In Figure 3, the resulting tensor has rank 1, which is the same as  $\vec{u}$ 's rank. However, the resulting tensor size can be much larger than the original tensors. Suppose we contract tensors of rank  $d_1$  and  $d_2$  which share a single common axis and each axis has dimension 2, then the size of the resulting tensor will be  $2^{d_1+d_2-2}$  and thus scales exponentially in tensor ranks.

### 2.3.2 Contraction order

Though we can carry out the contraction of a TN in any order, the size of the TN in intermediate steps of the contraction can vary widely. Ideally, a contraction will choose an order that limits the memory required to store the TN during all steps of the contraction, making it feasible. Given a contraction order, we can define the contraction width  $W$  of the TN [38] in two equivalent ways:

$$W = \begin{cases} \max_{v \in P} \text{deg}(v) & \text{(graphical)}, \\ \max_{T \in \mathcal{T}} \log_2 s(T) & \text{(tensors)}. \end{cases} \quad (6)$$

For the graphical representation,  $P$  is the set of nodes representing the tensors present at any stage of the contraction. In the tensor representation,  $\mathcal{T}$  is the set of all tensors that are present at any stage of the contraction, and  $s(T)$  is the size (number of elements) of the tensor  $T$ . Note that  $W$  depends on the TN and the contraction order. Then, up to a prefactor [38],  $2^W$  captures the memory requirements for the entire contraction. We use the contraction width as a proxy to runtime because it defines the largest tensor that one must manipulate during the contraction using multilinear operations, which take polynomial time in the size of that tensor [38]. Finding such orderings is an optimization problem and algorithms exist to find optimized contraction ordering according to the TN structure. While finding the optimal contraction order is easy in some cases, for example, a square lattice, it is much more complex in others, such as random networks [30]. In general, the computational demands of a TN contraction grow exponentially with the number of tensors in both time and memory. Even so, a method called bond compression allows us to further optimize the contraction by accepting a little error. We review this method below, and we explain in Section 2.2.4 how we use bond compression in a novel way.

### 2.3.3 Bond compression

Bond compression involves, in its simplest form, performing a contraction-decomposition operation on adjacent tensors within the TN. The term "bond" refers to the common index between tensors. The decomposition step primarily uses rank-revealing methods such as QR or singular value decomposition (SVD). Of these, the SVD plays a central role in TN algorithms. By setting a threshold value for singular values, either absolute or relative, we retain only the

<sup>1</sup> Though this does not factor into our work, it is also possible to have "free" edges with only one end connected to a node, indicating an axis in which no tensor multiplication occurs.

singular values above the threshold and corresponding singular vectors, thereby approximating subsequent contractions. This approach facilitates the contraction of larger TNs by reducing the contraction width during the process. However, in general, this comes at the expense of approximating the final result.

We implement bond compression as follows. Given two adjacent tensors  $T_A$  and  $T_B$  in the network, we transform them into the approximate tensors  $\tilde{T}_A$  and  $\tilde{T}_B$  as

$$\begin{aligned}
 T_A T_B &= Q_A R_A R_B Q_B \\
 &= Q_A R_{AB} Q_B \\
 &= Q_A (U \Sigma V^\dagger) Q_B \\
 &\approx Q_A (\tilde{U} \tilde{\Sigma} \tilde{V}^\dagger) Q_B \\
 &= (Q_A \tilde{U} \tilde{\Sigma}^{\frac{1}{2}}) (\tilde{\Sigma}^{\frac{1}{2}} \tilde{V}^\dagger Q_B) \\
 &= \tilde{T}_A \tilde{T}_B.
 \end{aligned}
 \tag{7}$$

The first equality comes after applying a QR decomposition to the tensors. Since the QR decomposition operates solely on matrices, we first need to reshape those tensors into matrices before decomposing them. Concretely, if we have a tensor  $T$  that has indices  $(i_1, i_2, \dots, i_k)$  and we want to apply the QR on the index  $i_3$ , then the reshaping would give a matrix with indices  $(\prod_{j \neq 3} i_j, i_3)$  (where the product signifies grouping the indices into a composite index). This matrix allows for the direct application of the QR decomposition on the desired dimension. The second equality comes from multiplying the matrices  $R_A$  and  $R_B$  to get the matrix  $R_{AB}$ . The third equality comes after performing the SVD on  $R_{AB}$ . Then, the threshold is applied, reducing the sizes of the singular values matrix, of  $U$  and of  $V$  and possibly approximating the result. The following equality comes from splitting this diagonal singular values matrix into two equal ones. The final equality comes from multiplying the matrices together in each parenthesis to get two new tensors with a “compressed” bond between them. This schedule optimizes the bond compression since the contraction between two tensors of possibly high dimensions is avoided.

### 3 Methodology

#### 3.1 Tensor networks for $p$ -XORSAT

As shown in Ref. [22], we can map any  $p$ -XORSAT instance as a TN. Contracting it will yield the number of solutions to the problem. As with the  $p$ -spin model in Section 2.1, we can define a  $p$ -XORSAT instance by a bipartite graph  $G = (U, V, E)$  and a vector  $\vec{b}$  of parities. Then, to each node  $u \in U$  we will assign a “variable” (or COPY) tensor, which has the form:

$$T_{i_1 i_2 \dots i_d}^{\text{COPY}\{u\}} = \begin{cases} 1, & \text{if } i_1 = i_2 = \dots = i_d, \\ 0, & \text{else,} \end{cases}
 \tag{8}$$

where the indices  $i_1 i_2 \dots i_d$  are boolean and  $d = \text{deg}(u)$ . For each node  $v \in V$ , we will assign a “clause” (or XOR) tensor of the form:

$$T_{i_1 i_2 \dots i_p}^{\text{XOR}\{v\}} = \begin{cases} 1, & \text{if } i_1 \oplus i_2 \oplus \dots \oplus i_p = b_v, \\ 0, & \text{else} \end{cases},
 \tag{9}$$

where the indices are also boolean,  $p = \text{deg}(v)$  and  $b_v$  is the parity associated to clause  $v$ . Finally, the edges  $E$  indicate which

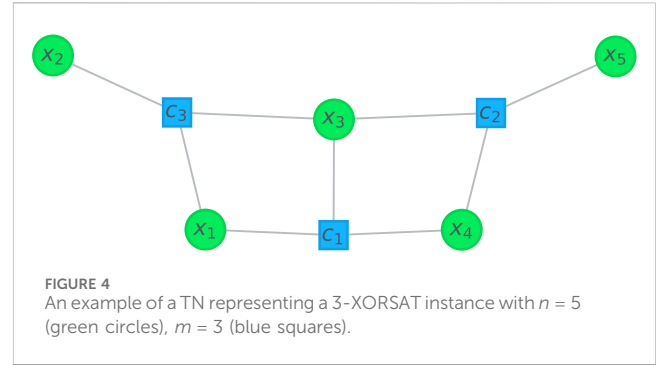


FIGURE 4 An example of a TN representing a 3-XORSAT instance with  $n = 5$  (green circles),  $m = 3$  (blue squares).

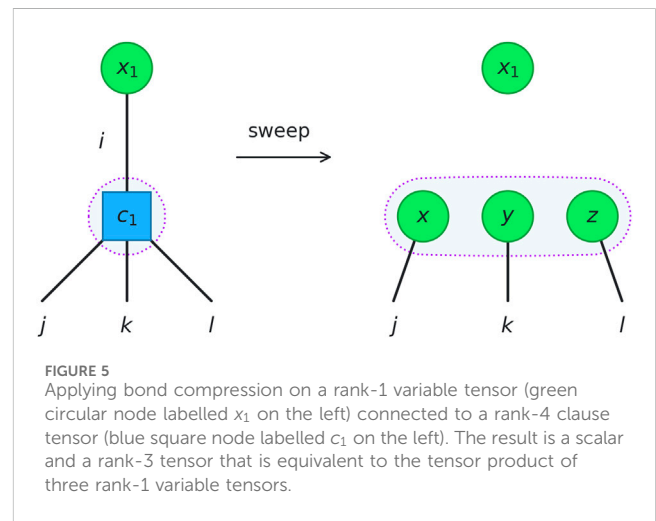


FIGURE 5 Applying bond compression on a rank-1 variable tensor (green circular node labelled  $x_1$  on the left) connected to a rank-4 clause tensor (blue square node labelled  $c_1$  on the left). The result is a scalar and a rank-3 tensor that is equivalent to the tensor product of three rank-1 variable tensors.

indices are common between different tensors in the TN and need to be summed over. Obtaining the solution count for the problem involves writing a summation over all of the common indices, yielding an expression similar (but much more involved for larger TNs) to Equation 5. In Figure 4, we give an example of a 3-XORSAT instance with  $n = |U| = 5$  and  $m = |V| = 3$  where the green circles represent tensors built following Equation 8 and the blue squares represent tensors built following Equation 9.

As explained in Section 2.3.2, we can evaluate the contraction width  $W$  of those TNs by extracting the highest tensor rank reached during its contraction.

#### 3.2 Eliminating redundancies through bond compression

There are several possible simplifications for a  $p$ -XORSAT problem that occur during the intermediate steps of the TN contraction. By recognizing these simplifications, we can reduce the size of the TN and therefore the time and memory requirements for its contraction. We will focus on the case where  $\vec{b} = \vec{0}$ , so all parities are even.

We will use bond compression to contract and decompose all adjacent tensors in the TN, a process commonly called a sweep, which is standard practice in TN methods. However, we will not

remove any nonzero singular values in the decomposition. If the tensors are full-rank, this is useless; the tensors remain unchanged after performing bond compression. On the other hand, TNs representing  $p$ -XORSAT problems often contain redundancy (see Section 2.2.4), which results in singular values that are zero to numerical accuracy. Therefore, performing bond compression and removing null singular values allows us to reduce the tensor sizes while keeping the resulting contraction exact.

An interesting fact with this method is that applying bond compression to the bond between a rank-1 variable tensor and a rank- $d$  clause tensor will effectively remove the bond, leading to a scalar (rank-0 tensor) and a rank- $(d-1)$  tensor. This rank- $(d-1)$  tensor will be composed of only ones (with a prefactor), which is equivalent to the tensor product of  $d-1$  rank-1 variable tensors. We illustrate an example of this in Figure 5. The following sweep step will then remove those  $d-1$  bonds (because they connect to a rank-1 variable, or COPY, tensor). This means the algorithm effectively removes the clause tensor and all its bonds, which is equivalent to one step in the leaf removal algorithm. This process could cascade through the entire TN, potentially eliminating all its bonds or resulting in a leafless core, giving the same outcome as the leaf removal algorithm. Therefore, bond compression sweeps automatically implement the leaf removal algorithm.

The contraction width will be the figure of merit for the performance of this algorithm because of its relation with the maximum intermediate tensor size (see Equation 6).

### 3.3 Graphical contraction

When  $\alpha < \alpha_d$ , leaf removal is likely to completely simplify the graph encoding the problem (Section 2.2.3). Translated to TN contraction, the bond compression shown in Figure 5 would be enough to dramatically simplify the TN contraction. This allows us to scale our simulations to large system sizes. However, when  $\alpha > \alpha_d$ , a core will likely remain. In this case, the remaining TN to contract comprises a core, and this will change the scaling of resources. In particular, the presence of a core will increase the contraction width (and therefore the memory requirements) much more quickly than when  $\alpha < \alpha_d$ . This limits our ability to test the performance of our algorithm on large instances in this regime.

To bypass this bottleneck and provide further scaling evidence, we develop a graphical algorithm that allows us to study the contraction width throughout a contraction by only studying the connectivity of the instance's graph. As discussed in Section 2.3.1, this is always possible for any exact contraction of a TN, since one simply needs to keep track of the tensor ranks at each step of the contraction (regardless of the tensors' contents). However, because we seek to study the performance of our TN algorithm that detects simplifications through bond compression, we must also encode the graphical patterns that will lead to simplifications. We will make use of the graphical simplifications discussed in Section 2.2.4, as well as more discussed in Section 3 of Ref. [35].

The graphical algorithm works as follows. Starting from a graph  $G$  encoding the instance, each node will always represent either a variable or a clause, and by default we will assign each node to a distinct "cluster". The algorithm "contracts" two nodes by assigning them to the same cluster. One can think of the cluster as a contracted

tensor. Then, whenever the algorithm performs a "sweep", it will search for any possible simplifications between clusters involving variable and clause nodes. If the algorithm finds any, it will perform the simplifications by removing edges in the problem<sup>2</sup>. The algorithm alternates between sweeping and contracting until every node in the graph belongs to the same cluster, in which case it terminates. It uses the same contraction ordering as in our TN algorithm. In graphical contraction, the goal is to obtain the sizes of intermediate tensors encountered in the contraction, not the values of the tensors themselves. Therefore, the graphical algorithm will not produce a solution count, just a contraction width. We also note that a degree-2 variable tensor is, in its tensor representation, equal to a  $2 \times 2$  identity matrix (see Equation 8). Knowing that, we can replace any degree-2 variable nodes in a cluster by edges.

The rank of an intermediate tensor is the number of outgoing edges from a cluster, and its size is:

$$\text{size}_{\text{cluster}} = 2^{\#\text{outgoing edges}}. \quad (10)$$

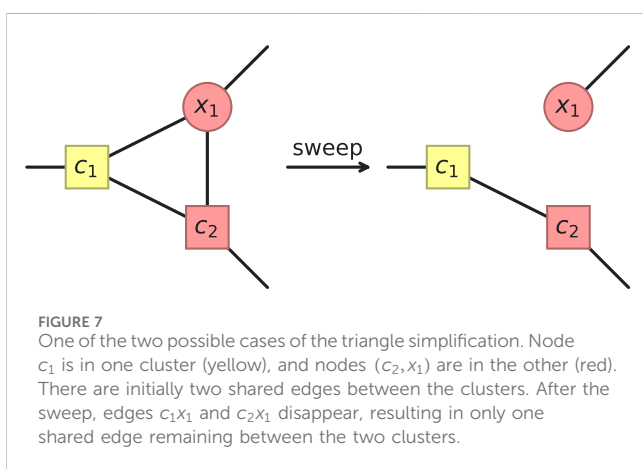
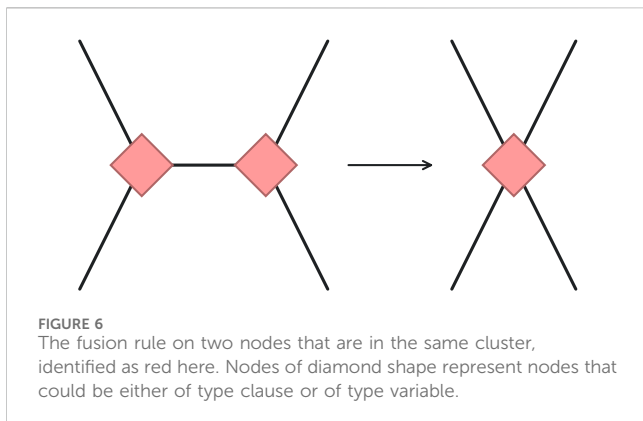
Taking the maximum number of outgoing edges over all contraction steps and clusters directly yields the contraction width.

We now interpret the sweeping method as implementing graphical simplifications. Recall that the TN contraction is a sum over all the boolean indices of the tensors and only the indices which satisfy the logic of the TN will contribute 1 to the sum (and 0 otherwise), yielding the solution count to the problem. Therefore, any simplifications from bond compression must correspond to redundancy in specifying the logic of the TN. Suppose the algorithm is compressing the bonds between tensors  $T_A$  and  $T_B$ . For concreteness, suppose there are  $k$  bonds. The algorithm will first transform the  $k$  bonds of dimension 2 into a single bond of dimension  $2^k$ . Then, the algorithm will compress that bond according to Equation 7, yielding new tensors  $\tilde{T}_A$  and  $\tilde{T}_B$  such that their shared bond is minimized due to the SVD. We observe that the new shared bond has dimension  $2^{k'}$  for  $k' \leq k$ , and  $k'$  corresponds to the minimum number of bits needed to preserve the logic of contracting  $T_A$  and  $T_B$ . Note that we can interpret a single bond of dimension  $2^{k'}$  as  $k'$  bonds of dimension 2, which is how we display our graphical simplifications.

For example, in the leaf removal algorithm, compressing the bond of a rank-1 variable tensor  $T^{\text{COPY}}$  with a rank-4 clause tensor  $T^{\text{XOR}}$  will yield a shared "bond" of dimension  $2^0$ , due to redundancy in the representation of contracting those two tensors. This dimension 1 "bond" signifies that the contraction of those tensors will be a tensor product that reduces to an element wise multiplication of tensor  $\tilde{T}^{\text{XOR}}$  with the scalar value of  $\tilde{T}^{\text{COPY}}$ . Similarly, we show below that there are several known logical simplifications present between tensors in these TNs which minimize the number of bits needed to preserve the contraction, implying the QR/SVD will find them. We observe as much in our experiments, which led us to developing our graphical algorithm.

The algorithm must detect and simplify any tensor that our TN algorithm would simplify. For the (2,3)-biregular graph ensemble ( $\alpha = 2/3$  leaf-free instances) we consider, only a

<sup>2</sup> The algorithm can also remove edges within a cluster, if it is part of the simplification (see Figure 7).

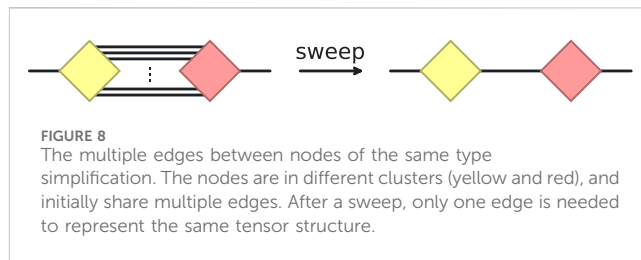


subset of the possible  $p$ -XORSAT simplifications are present. Following the examples in Ref. [35], our graphical algorithm detects the following possible simplifications (we assume  $\vec{b} = \vec{0}$  for simplicity):

- Fusion rule,
- Generalized Hopf law,
- Triangle simplification,
- Multiple edges between nodes of the same type,
- Scalar decomposition.

The fusion rule says that neighboring clause nodes in the same cluster can be contracted together to form a bigger clause node, and the same is true for variable nodes. In this case, we actually replace the two nodes with a single node representing them. Their corresponding tensor representations would then be exactly those of a clause or variable tensor of larger rank. This rule is schematically shown in Figure 6. One can also apply the same rule for nodes of the same type which share multiple edges. However, for clause nodes, there will be an overall numerical factor of  $2^{\#\text{shared edges}-1}$  in the entries of the tensor, corresponding to the summation over shared indices. Since we are only concerned with the size of the tensors, this coefficient is not relevant.

The generalized Hopf law ensures that if a clause node and a variable node share  $t$  edges and the degree of each is greater than  $t$ , a



sweep will leave  $t \bmod 2$  edges between them (as discussed in Section 2.2.4).

The triangle simplification is an implementation of the Hopf law between two clusters that, between them, contain a “triangle” of nodes. Those triangles contain two nodes of one type (clause or variable) and one of the other. Because we always contract nodes of the same type within a cluster using the fusion rule, a triangle simplification can only occur when the nodes of the same type are in different clusters. When we sweep between these clusters, applying the fusion rule and then a basic Hopf law will remove edges, as shown in Figure 7.

The simplification of multiple edges between nodes of the same type is a variant of the fusion rule. Consider the example in Figure 8. If the nodes are in different clusters, sweeping would not contract the nodes, but would simplify all the edges except one in the same way as a the fusion rule (ignoring once again an overall factor).

Finally, the scalar decomposition occurs when there are two nodes of the same type and at least one shares all its edges with the other. A sweep will merge the two nodes, and then only factor out a scalar (degree-0 node) in the decomposition to return to two tensors. However, the sweep will remove all edges between the tensors.

We now argue that these simplifications are sufficient to characterize any possible simplification present in the (2,3)-biregular graph ensemble. Each variable node has degree 2, so the bialgebra law and any higher-order generalizations cannot occur because they require variable nodes of degree at least 3. Because we replace any degree-2 variable node in a cluster by an edge and the fusion rule combines clause nodes within a cluster, most clusters will be a single clause node of some degree. Our rules above capture simplifications between such clusters. The one exception is that variable nodes are their own clusters at the start of the algorithm before being contracted with other nodes. In this case, the simplifications given by Figure 7 may apply. Therefore, our set of graphical rules should be sufficient to capture all possible simplifications in this ensemble. We also provide evidence of this claim in Section 4.2.

## 3.4 Numerical experiments and tools

### 3.4.1 Generation of random instances

To generate our instances at a given  $\alpha$  and  $n$ , we choose  $m = \alpha n$  tuples<sup>3</sup> of  $p$  variables uniformly at random without replacement

<sup>3</sup> Note that we choose  $m, n$ , and  $\alpha$  such that  $m$  and  $n$  are integers.

from  $\{x\}$ , the set of variables defined in Section 2.2. This means that each variable tensor's rank  $d$  conforms to the following Poisson distribution:

$$\mathcal{P}(\text{rank}(x_i) = d) = \frac{(p\alpha)^d}{d!} e^{-p\alpha}.$$

This rank is defined as the number of times that a variable is present in the problem. In the language of Equation 2, we randomly place  $p$  ones in each row of  $A$  and the rank of the variable  $x_i$  corresponds to the number of ones in column  $i$ . For our numerical experiments, we set  $p = 3$ . We also exclusively focus on the case  $\vec{b} = \vec{0}$  (the unfrustrated version of the  $p$ -spin model). We do so because in the regime  $\alpha < \alpha_c$  that we study, the problem will contain at least one solution for any given  $\vec{b}$  (with high probability in the limit of large problem size), which allows us to redefine the problem such that  $\vec{b} = \vec{0}$  [5, 34] and the solution count remains the same.

### 3.4.2 Generation of leaf-free instances

Since we are mainly concerned with the scaling of resources for instances which contain a core, we choose a minimal ensemble with this property. We will study the ensemble of connected 3-regular graphs on  $m$  clause nodes generated uniformly at random using the Degree\_Sequence function in `igraph` with the Viger-Latapy method [39]. To create a 3-XORSAT instance, we place a variable node along each edge of the regular graph. This ensures the variable nodes all have degree two, and the clause nodes have degree three. Therefore, the ensemble of instances is for  $\alpha = 2/3$ . Note that this is below  $\alpha_d$ , but the method of construction explicitly ensures a core.

### 3.4.3 Implementation of contraction methods

For TN contractions, we use `quimb`, a Python package for manipulating TNs [40]. For the graphical method, we use `igraph`, an efficient network analysis library [41], in order to work with node attributes on the graph directly. Those attributes let us define the node types (clause and variable) and the nodes' clusters.

The TN contraction order, as discussed in Section 2.3.2, determines the contraction width. Without applying our sweeping method, one can track this quantity without actually performing the tensor contraction. One must simply keep track of the ranks of the tensors at any point in the contraction, noting as in Section 2.3.1 that combining two tensors yields a new tensor of known rank. We use `cotengra`, a Python package for TN contractions, to track this quantity [38]. In order to track this quantity when sweeps are applied, we use `quimb` in order to read the tensors' sizes during the contraction and calculate the contraction width using Equation 6.

For random TNs such as ours, there exist multiple heuristic algorithms for finding contraction orderings [30, 38] which lower the contraction width and are practically useful for carrying out computations. For the results in Section 4, we determine the ordering using a community detection algorithm based on the edge betweenness centrality [42] (EBC) of the network. This algorithm is implemented as `community_edge_betweenness` in the Python package `igraph` [41]. We use the EBC algorithm because it looks for communities in the graph, thus contracting dense sections first. This is useful in random TNs because it minimizes the chances of having to work with huge tensors quickly, which could result in a tensor of large

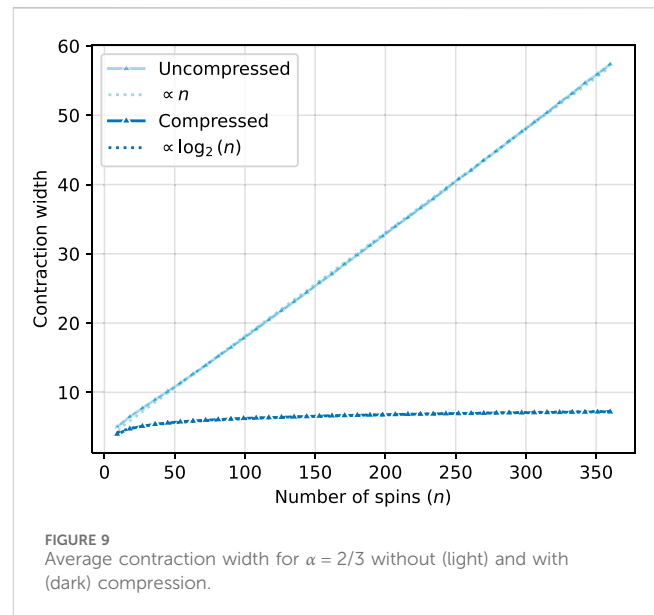


FIGURE 9  
Average contraction width for  $\alpha = 2/3$  without (light) and with (dark) compression.

rank (and therefore, large contraction width). This algorithm is also deterministic, ensuring reproducibility of the contraction orderings. Furthermore, in Section 4.2, we compare the results obtained using this contraction ordering with two others: `KaHyPar` [43, 44] and `greedy`, both from the Python package `cotengra`.

Even with these better contraction orderings, exactly contracting these random TNs without bond compression will generally result in an exponential growth in  $n$  of time and memory (see Section 4). However, we will show that by manipulating the TN after each contraction using the algorithm defined in Section 3.4.4, we can alter the scaling of resources for a range of parameter values in the problem.

### 3.4.4 Sweeping method

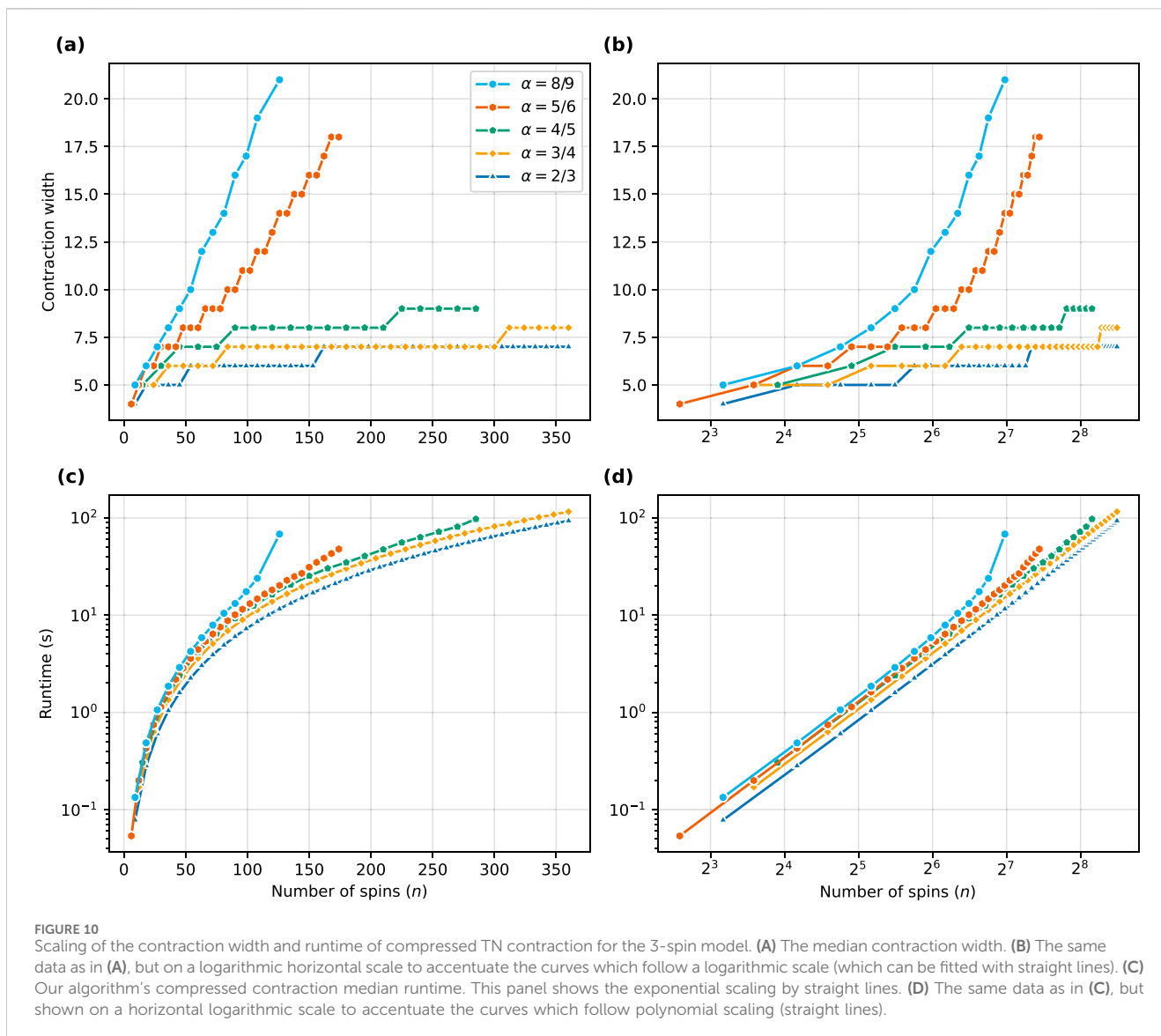
To ensure lossless compression in bond sweeping, we set the relative threshold for zero singular values to be  $10^{-12}$ . We sweep the TN in arbitrary order until the tensor sizes converge. During a sweep, we compress all the bonds using the `compress_all` method implemented in `quimb`, which uses the compression schedule described in Equation 7. We perform sweeps before each contraction, potentially finding simplifications (see Section 2.2.4) in the structure of the TN during each step of the full contraction.

## 4 Results

### 4.1 Numerical contraction for random instances

Numerical TN contractions were performed on an AMD EPYC 7F72 @ 3.2 GHz processor, with a maximum allocated RAM of 1 TB. Each point in the figures of this section corresponds to the median contraction width or contraction runtime over  $10^4$  instances for a given number of spins  $n$ , except for Figure 9 which shows the average scaling of the contraction width. The contraction width determines, to leading order, the contraction runtime. As is common in random





graph ensembles for spin-glass models or Boolean variable graphical models, the instance samples contains outliers that are much harder to solve than the typical instance.

In Figure 9, we show the average contraction width with and without compression (sweeping) for  $\alpha = 2/3$ . Without compression, the scaling of the average contraction width is linear, indicating exponential growth of tensor sizes. By contrast, compression changes the scaling to one that is well described by a logarithmic curve, indicating polynomially growing tensor sizes and hence contraction runtimes.

We studied larger values of  $\alpha$  and we show in Figure 10 how the scaling of both the median contraction width and median contraction runtime evolve as  $\alpha$  increases. For the largest system sizes, out of  $10^4$  instances, a few outliers require times beyond any reasonable timeout we have tried, as expected. We therefore cannot report unbiased runtime averages for these sizes. However, when plotted against system size, the data for the average and median of the contraction width are comparable (likewise for the contraction runtime). Note that we observe that the average contraction width is

a smoother function of system size than the median contraction width; though we show the median contraction width in Figure 10, we use the average contraction width data to extract a scaling. We do the same for the contraction runtime. In this case, the curves in the bottom of Figure 10 are already smooth.

The results in Figure 10A highlight linear scaling of the curves for  $\alpha = 5/6$  and  $\alpha = 8/9$  while Figure 10B clearly shows the logarithmic nature of the curves for  $\alpha = 2/3$  and  $\alpha = 3/4$ . For  $\alpha = 4/5$ , this median scaling seems to be of logarithmic nature in Figure 10B, but analysing the average shows that it actually starts to “peel-off” from logarithmic scaling.

The logarithmic scaling for  $\alpha = 2/3$  and  $\alpha = 3/4$  is mainly due to the TN algorithm automatically implementing leaf removal, since  $\alpha < \alpha_d$ . Indeed, this leads to a high probability that the initial sweeps will remove all the edges in the TN even before the first contraction, leaving only scalars to be multiplied. For  $\alpha = 5/6$  and  $\alpha = 8/9$ , values that are greater than  $\alpha_d$ , we find that the algorithm is less efficient due to a core that remains after the initial sweeps. Those cores lead to actual tensor contractions instead of scalar multiplications, so the

TABLE 1 Performance comparison between optimized compressed TN contraction and GE.

Methods	$\alpha$	Memory	Time
Standard GE	2/3	$\propto n$	$\propto n$
Standard GE	3/4	$\propto n^2$	$\propto n^3$
Smart GE	$< \alpha_d$	$\propto n$	$\propto n$
Compressed TN	2/3	$\propto n^{1.030}$	$\propto n^{1.226}$
Compressed TN	3/4	$\propto n^{1.112}$	$\propto n^{1.420}$

instances with  $\alpha = 5/6$  and  $\alpha = 8/9$  become harder to compute, hence the contraction widths' polynomial scaling. As we noted, since  $\alpha = 4/5$  is close to  $\alpha_d$ , there is a probability of a core remaining for our finite system sizes, so the algorithm starts becoming less efficient here too. Sweeping still removes all the edges in the TN in most cases, but less so than with  $\alpha = 3/4$ , thus the “peel-off” starting at  $\alpha = 4/5$ .

In Figure 10C, we see the scaling of the median contraction runtime (in seconds) with a logarithmic vertical axis and the same data is shown with a logarithmic horizontal axis in Figure 10D. Accordingly with the contraction width scaling, we find polynomial curves for  $\alpha = 2/3$  ( $\propto n^{1.928}$ ) and  $\alpha = 3/4$  ( $\propto n^{1.902}$ ). For smaller  $n$ , we see that the time scaling for all the curves follow a polynomial scaling. This is due to the small finite size of the TN, since it changes for bigger TNs, or for larger  $n$ . The “peel-off” phenomenon is thus also observed at the end of the curves for  $\alpha \in \{4/5, 5/6, 8/9\}$ , becoming more pronounced with increasing  $\alpha$ . This means that the scaling transitions from polynomial to superpolynomial, like the conclusion on memory usage in Figure 10B.

At  $\alpha = 2/3$  and  $3/4$ , the compressed TN algorithm exhibits performance between those of the standard and “smart” GE methods (see Table 1). For these values of the  $\alpha$  parameter, the contraction runtime can be further improved by removing bonds of dimension 1 after each contraction step. Indeed, when a bond is completely compressed by our algorithm, a dimension 1 bond remains between the two neighboring tensors. These dimension 1 bonds do not affect memory scaling, yet the sweeping algorithm will continue trying to compress them, even though they cannot be further compressed. Eliminating those “useless” bonds results in improved polynomial contraction runtimes for  $\alpha = 2/3$  and  $3/4$ , as shown in Table 1, since the subsequent sweeps will not try to compress those bonds anymore. In this same table, the memory is defined as the maximum size of the whole TN—the sum of all its tensors' sizes—reached during its contraction with the sweeping method applied.

## 4.2 Graphical contraction for leaf-free instances

For the leaf-free ensemble, each point in the figures has been averaged over 200 random leaf-free instances. With the graphical method, the contraction widths are extracted from the number of clusters' outgoing edges during the TN contraction, as explained in Section 3.3 (see Equation 10). All the results for the contraction

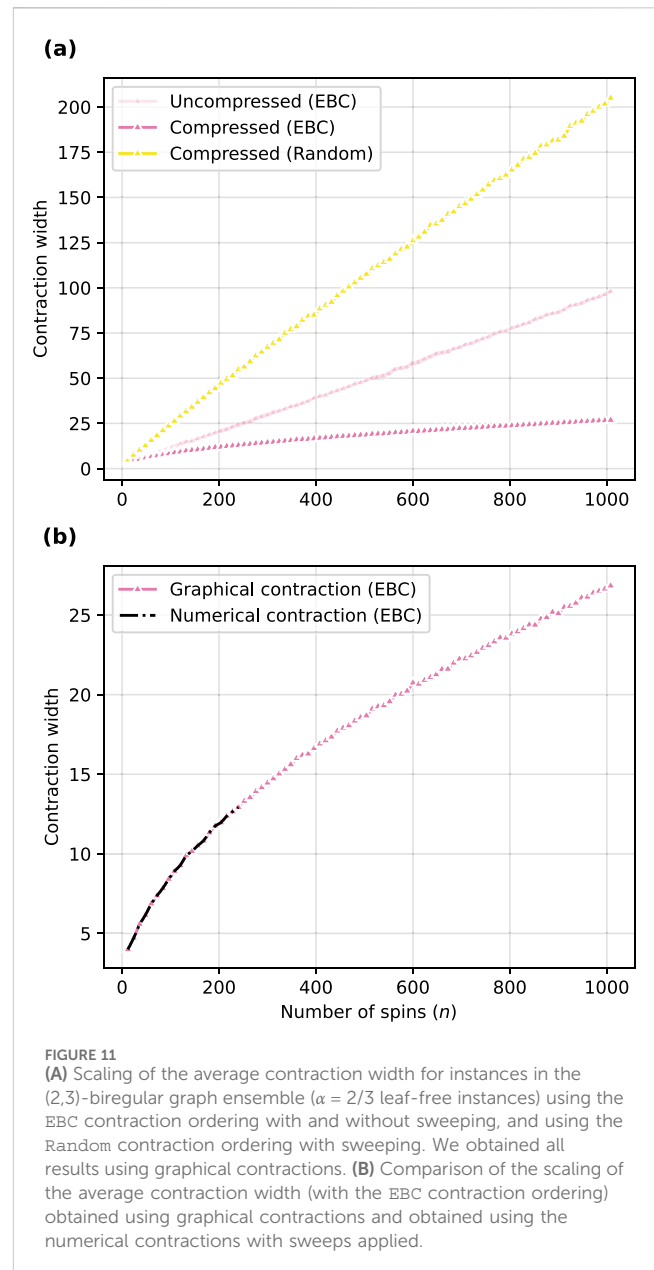


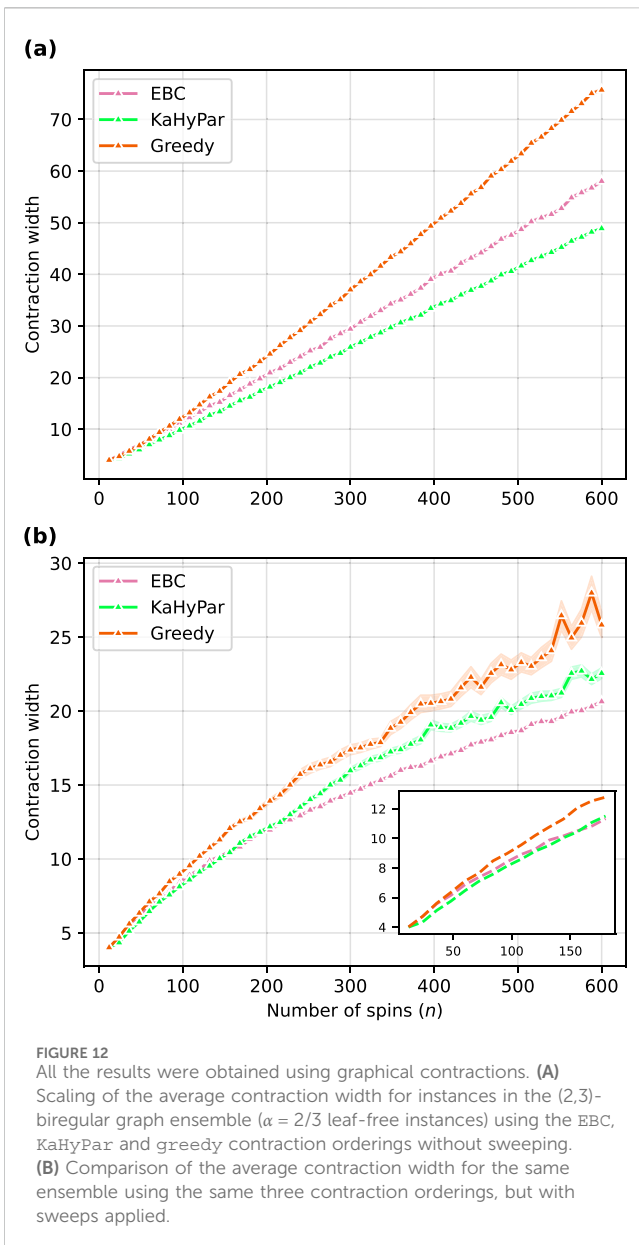
FIGURE 11

(A) Scaling of the average contraction width for instances in the (2,3)-biregular graph ensemble ( $\alpha = 2/3$  leaf-free instances) using the EBC contraction ordering with and without sweeping, and using the Random contraction ordering with sweeping. We obtained all results using graphical contractions. (B) Comparison of the scaling of the average contraction width (with the EBC contraction ordering) obtained using graphical contractions and obtained using the numerical contractions with sweeps applied.

width obtained with this graphical method are shown in Figures 11, 12.

Now having the possibility to study larger TNs without being limited by the memory, we can compare the contraction width of the algorithm on different contraction orderings. In Figure 11A, we compare two of them: EBC and Random. The Random method chooses the next tensors to be contracted completely randomly. It can thus only be usefully studied with this graphical method because it quickly scales to astronomical contraction widths, as seen in Figure 11A.

From Figure 11A, we see that a good contraction ordering is an important factor for the success of the sweeping method during the contraction of a given TN that models a  $p$ -XORSAT problem. Two known methods for random tensor networks have also been used in order to compare the results obtained from the EBC method, as shown in Figure 12.



The results demonstrate that the sweeping method finds enough simplifications for instances in the (2,3)-biregular graph ensemble so that the scaling of the average contraction width changes from linear to sublinear for the EBC, KaHyPar and greedy contraction orderings. From those results, after  $n \approx 150$ , we see that the EBC method is most efficient in finding those simplifications of the three, followed by KaHyPar and then greedy. The precise functional form of the scaling is nontrivial and we have not been able to determine a sufficiently accurate fitting function. This means that the sweeping method goes beyond the efficacy of the leaf removal in the TN representation of the 3-spin model.

Note that the (2,3)-biregular graph ensemble we consider offers a simplification of the corresponding  $p$ -XORSAT problem which allows leaf removal—and by extension, our TN algorithm—to work efficiently in polynomial time. Suppose  $A$  is the  $m \times n$  matrix encoding the problem. By definition of the ensemble, each column of  $A$  has exactly two 1s. Therefore, the rows of  $A$  satisfy

$A_1 + A_2 + \dots + A_m = 0 \pmod{2}$ , indicating the rows are linearly dependent. In other words, we can remove some row  $A_i$  from the problem without changing the solution space and count. In terms of the graph, one can remove the corresponding clause node encoding  $A_i$  because it is made redundant by the other clauses. However, removing a clause node allows leaf removal to begin since the variable nodes that were connected to that removed clause node will now be degree-1. Leaf removal will then succeed in solving the problem and producing an empty core, implying both leaf removal and our TN algorithm are efficient for this ensemble if we first remove a single redundant clause.

We have verified that the graphical contraction method of Section 3.3 yields tensor sizes identical to those found via numerical contraction at each contraction step by comparing the two methods for 100 random instances with  $n = 81$  (for the EBC and Random contraction orderings). Moreover, all contraction widths for the 200 random instances used to get the results in Figure 11B with sizes up to  $n = 240$  are identical to those obtained with numerical contraction (for the EBC contraction ordering).

## 5 Conclusion

In this work, we have applied compressed TN contraction to the  $p$ -spin model. Focusing on  $p = 3$ , we have shown that lossless compression sweeps over the bonds of the network emulate the leaf removal algorithm, meaning that the TN method is efficient (i.e., polynomial-time) below the dynamical transition at  $\alpha_d \approx 0.818$ . Above the dynamical transition, the appearance of a leafless core adversely affects the performance of the TN algorithm, which is now superpolynomial-time. Nevertheless, by focusing on the restricted ensemble of biregular instances where every spin participates in exactly two interactions, we find that compressed contraction can be done in subexponential time. This speedup over the anticipated exponential scaling depends crucially on the choice of contraction path. We note that, unlike some previous TN techniques applied to spin-glass models [45], our methods are exact and can be made to suffer no loss of precision for the case of XOR constraints. Indeed, we observe that the local singular values during each sweeping step correspond to either positive or fractional powers of two if they are distributed properly after having applied the SVD. This means that we either have those values or zero/numerical zero singular values. A similar observation has been made for Clifford circuits, essentially parity circuits, where stabilizer states possess flat entanglement spectra [46–48]. To our knowledge, this is the first general-purpose numerical method for spin-model partition function and model counting computations that achieves this performance for  $p$ -spin models without invoking GE as a subroutine. Furthermore, we believe that this is the first nontrivial case of a spin model defined on random sparse graphs (that are not trees) where compressed TN contraction solves the model exactly, yet leads to an exponential-to-subexponential speedup over direct TN contraction.

## Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

## Author contributions

BL: Conceptualization, Data curation, Formal Analysis, Investigation, Methodology, Software, Validation, Visualization, Writing—original draft, Writing—review and editing. JC: Conceptualization, Formal Analysis, Investigation, Methodology, Supervision, Validation, Writing—original draft, Writing—review and editing. SK: Conceptualization, Formal Analysis, Funding acquisition, Methodology, Project administration, Resources, Software, Supervision, Writing—original draft, Writing—review and editing.

## Funding

The author(s) declare that financial support was received for the research, authorship, and/or publication of this article. This work was supported by the Ministère de l'Économie, de l'Innovation et de l'Énergie du Québec through its Research Chair in Quantum Computing, an NSERC Discovery grant, and the Canada First

Research Excellence Fund. This work made use of the compute infrastructure of Calcul Québec and the Digital Research Alliance of Canada.

## Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

## References

- Kirkpatrick S, Toulouse G. Configuration space analysis of travelling salesman problems. *J Phys France* (1985) 46:1277–92. doi:10.1051/jphys:019850046080127700
- Bryngelson JD, Wolynes PG. Spin glasses and the statistical mechanics of protein folding. *Proc Natl Acad Sci* (1987) 84:7524–8. doi:10.1073/pnas.84.21.7524
- Venkataraman G, Athithan G. Spin glass, the travelling salesman problem, neural networks and all that. *Pramana* (1991) 36:1–77. doi:10.1007/BF02846491
- Stein DL, Newman CM. *Spin glasses and complexity*. Princeton University Press (2013). doi:10.23943/princeton/9780691147338.001.0001
- Mézard M, Ricci-Tersenghi F, Zecchina R. Two solutions to diluted p-spin models and XORSAT problems. *J Stat Phys* (2003) 111:505–33. doi:10.1023/A:1022886412117
- Ricci-Tersenghi F. Being glassy without being hard to solve. *Science* (2010) 330:1639–40. doi:10.1126/science.1189804
- Bernaschi M, Bisson M, Fatica M, Marinari E, Martin-Mayor V, Parisi G, et al. How we are leading a 3-xorsat challenge: from the energy landscape to the algorithm and its efficient implementation on gpus(a). *Europhysics Lett* (2021) 133:60005. doi:10.1209/0295-5075/133/60005
- Kanao T, Goto H. Simulated bifurcation for higher-order cost functions. *Appl Phys Express* (2022) 16:014501. doi:10.35848/1882-0786/acaba9
- Aadit NA, Nikhar S, Kannan S, Chowdhury S, Camsari KY. All-to-all reconfigurability with sparse Ising machines: the XORSAT challenge with p-bits (2023). doi:10.1088/arXiv:2312.08748
- Jörg T, Krzakala F, Semerjian G, Zamponi F. First-order transitions and the performance of quantum algorithms in random optimization problems. *Phys Rev Lett* (2010) 104:207206. doi:10.1103/PhysRevLett.104.207206
- Farhi E, Gosset D, Hen I, Sandvik AW, Shor P, Young AP, et al. Performance of the quantum adiabatic algorithm on random instances of two optimization problems on regular hypergraphs. *Phys Rev A* (2012) 86:052334. doi:10.1103/PhysRevA.86.052334
- Hen I. Equation planting: a tool for benchmarking ising machines. *Phys Rev Appl* (2019) 12:011003. doi:10.1103/PhysRevApplied.12.011003
- Bellitti M, Ricci-Tersenghi F, Scardicchio A. Entropic barriers as a reason for hardness in both classical and quantum algorithms. *Phys Rev Res* (2021) 3:043015. doi:10.1103/PhysRevResearch.3.043015
- Kowalsky M, Albash T, Hen I, Lidar DA. 3-regular three-xorsat planted solutions benchmark of classical and quantum heuristic optimizers. *Quan Sci Technology* (2022) 7:025008. doi:10.1088/2058-9565/ac4d1b
- Patil P, Kourtis S, Chamon C, Mucciolo ER, Ruckenstein AE. Obstacles to quantum annealing in a planar embedding of XORSAT. *Phys Rev B* (2019) 100:054435. doi:10.1103/PhysRevB.100.054435
- Haanpää H, Järvisalo M, Kaski P, Niemelä I. Hard satisfiable clause sets for benchmarking equivalence reasoning techniques. *J Satisfiability, Boolean Model Comput* (2006) 2:27–46. doi:10.3233/SAT190015
- Järvisalo M. Further investigations into regular xorsat. In: *Aaai* (2006). p. 1873–4.
- Jia H, Moore C, Selman B. From spin glasses to hard satisfiable formulas. In: Hoos HH, Mitchell DG, editors. *Theory and applications of satisfiability testing*. Berlin, Heidelberg: Springer Berlin Heidelberg (2005). p. 199–210.
- Barthel W, Hartmann AK, Leone M, Ricci-Tersenghi F, Weigt M, Zecchina R. Hiding solutions in random satisfiability problems: a statistical mechanics approach. *Phys Rev Lett* (2002) 88:188701. doi:10.1103/PhysRevLett.88.188701
- Ricci-Tersenghi F, Weigt M, Zecchina R. Simplest randomK-satisfiability problem. *Phys Rev E* (2001) 63:026702. doi:10.1103/PhysRevE.63.026702
- Guidetti M, Young AP. Complexity of several constraint-satisfaction problems using the heuristic classical algorithm walksat. *Phys Rev E* (2011) 84:011102. doi:10.1103/PhysRevE.84.011102
- Garcia-Saez A, Latorre JI. An exact tensor network for the 3SAT problem. (2011).
- Biamonte JD, Morton J, Turner J. Tensor network contractions for #sat. *J Stat Phys* (2015) 160:1389–404. doi:10.1007/s10955-015-1276-z
- Kourtis S, Chamon C, Mucciolo ER, Ruckenstein AE. Fast counting with tensor networks. *Scipost Phys* (2019) 7:060. doi:10.21468/SciPostPhys.7.5.060
- Meichanetzidis K, Kourtis S. Evaluating the jones polynomial with tensor networks. *Phys Rev E* (2019) 100:033303. doi:10.1103/PhysRevE.100.033303
- de Beudrap N, Kissinger A, Meichanetzidis K. Tensor network rewriting strategies for satisfiability and counting. *EPTCS* (2021) 340:46–59. doi:10.4204/eptcs.340.3
- Schuch N, Wolf MM, Verstraete F, Cirac JI. Computational complexity of projected entangled pair states. *Phys Rev Lett* (2007) 98:140506. doi:10.1103/PhysRevLett.98.140506
- Evenbly G, Vidal G. Tensor network renormalization. *Phys Rev Lett* (2015) 115:180405. doi:10.1103/PhysRevLett.115.180405
- Evenbly G. Algorithms for tensor network renormalization. *Phys Rev B* (2017) 95:045117. doi:10.1103/PhysRevB.95.045117
- Gray J, Chan GK-L. Hyperoptimized approximate contraction of tensor networks with arbitrary geometry. *Phys Rev X* (2024) 14:011009. doi:10.1103/PhysRevX.14.011009
- Alkabetz R, Arad I. Tensor networks contraction and the belief propagation algorithm. *Phys Rev Res* (2021) 3:023073. doi:10.1103/PhysRevResearch.3.023073
- Pancotti N, Gray J. One-step replica symmetry breaking in the language of tensor networks. (2023).
- Garey MR, Johnson DS. *Computers and intractability: a guide to the theory of NP-completeness*. San Francisco, CA: W. H. Freeman (1979).
- Braunstein A, Leone M, Ricci-Tersenghi F, Zecchina R. Complexity transitions in global algorithms for sparse linear systems over finite fields. *J Phys A: Math Gen* (2002) 35:7559–74. doi:10.1088/0305-4470/35/35/301
- Denny SJ, Biamonte JD, Jaksch D, Clark SR. Algebraically contractible topological tensor network states. *J Phys A: Math Theor* (2011) 45:015309. doi:10.1088/1751-8113/45/1/015309

36. Seitz P, Medina I, Cruz E, Huang Q, Mendl CB. Simulating quantum circuits using tree tensor networks. *Quantum* (2023) 7:964. doi:10.22331/q-2023-03-30-964
37. Wang M, Pan Y, Xu Z, Yang X, Li G, Cichocki A. Tensor networks meet neural networks: a survey and future perspectives. *arXiv preprint arXiv:2302.09019* (2023). doi:10.48550/arXiv.2302.09019
38. Gray J, Kourtis S. Hyper-optimized tensor network contraction. *Quantum* (2021) 5:410. doi:10.22331/q-2021-03-15-410
39. Viger F, Latapy M. Efficient and simple generation of random simple connected graphs with prescribed degree sequence. *J Complex Networks* (2015) 4:15–37. doi:10.1093/comnet/cnv013
40. Gray J. quimb: a python package for quantum information and many-body calculations. *J Open Source Softw* (2018) 3:819. doi:10.21105/joss.00819
41. Csardi G, Nepusz T. The igraph software package for complex network research. *InterJournal Complex Syst* (2006) 1695.
42. Girvan M, Newman MEJ. Community structure in social and biological networks. *Proc Natl Acad Sci* (2002) 99:7821–6. doi:10.1073/pnas.122653799
43. Schlag S, Henne V, Heuer T, Meyerhenke H, Sanders P, Schulz C. (????). *italicjkj/italicj-way Hypergraph Partitioning via italicjn/italicj-Level Recursive Bisection* 53–67. doi:10.1137/1.9781611974317.5
44. Akhremtsev Y, Heuer T, Sanders P, Schlag S. Engineering a direct *italicjkj/italicj-way Hypergraph Partitioning Algorithm*. In: 2017 Proceedings of the Nineteenth Workshop on Algorithm Engineering and Experiments (ALENEX), 28–42 (2017). doi:10.1137/1.9781611974768.3
45. Zhu Z, Katzgraber HG. Do tensor renormalization group methods work for frustrated spin systems? *arXiv preprint arXiv:1903.07721* (2019). doi:10.48550/arXiv.1903.07721
46. Fattal D, Cubitt TS, Yamamoto Y, Bravyi S, Chuang IL. Entanglement in the stabilizer formalism. *arXiv* (2004). doi:10.48550/arXiv.quant-ph/0406168
47. Hamma A, Ionicioiu R, Zanardi P. Bipartite entanglement and entropic boundary law in lattice spin systems. *Phys Rev A* (2005) 71:022315. doi:10.1103/PhysRevA.71.022315
48. Zhou S, Yang Z-C, Hamma A, Chamon C. Single T gate in a Clifford circuit drives transition to universal entanglement spectrum statistics. *Scipost Phys* (2020) 9:087. doi:10.21468/SciPostPhys.9.6.087