Hyperoptimized Approximate Contraction of Tensor Networks with Arbitrary Geometry

Johnnie Gray and Garnet Kin-Lic Chan
Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, California 91125, USA

(Received 27 July 2022; accepted 7 December 2023; published 26 January 2024)

Tensor network contraction is central to problems ranging from many-body physics to computer science. We describe how to approximate tensor network contraction through bond compression on arbitrary graphs. In particular, we introduce a hyperoptimization over the compression and contraction strategy itself to minimize error and cost. We demonstrate that our protocol outperforms both handcrafted contraction strategies in the literature as well as recently proposed general contraction algorithms on a variety of synthetic and physical problems on regular lattices and random regular graphs. We further showcase the power of the approach by demonstrating approximate contraction of tensor networks for frustrated three-dimensional lattice partition functions, dimer counting on random regular graphs, and to access the hardness transition of random tensor network models, in graphs with many thousands of tensors.

DOI: 10.1103/PhysRevX.14.011009

Subject Areas: Computational Physics, Quantum Physics, Statistical Physics

I. INTRODUCTION

Tensor network contraction, a summation over a product of multidimensional quantities, is a common computational structure. For example, this computation underlies quantum circuit simulation [1–9], quantum many-body simulations [10–17], evaluating classical partition functions [18–25], decoding quantum error correcting codes [26–32], counting solutions of satisfiability problems [25,33–39], statistical encoding of natural language [40–43], and many other applications. The cost of exact contraction scales, in general, exponentially with the number of tensors. However, there is evidence, for example, in some many-body physics applications, that tensor networks of interest can often be approximately contracted with satisfactory and controllable accuracy, without necessarily incurring exponential cost [44,45]. Many different approximation strategies for tensor network contraction have been proposed [12,13,17,20,23,46–50]. Especially in many-body physics contexts, the approximate contraction algorithms are usually tied to the geometry of a structured lattice. In this work, we consider how to search for an optimal approximate tensor network contraction strategy, within an approach that can be used not only for structured lattices, but also for arbitrary graphs. We view the essential prescription as the order in which contractions and approximate compressions are performed: this sequence can be summarized as a computational tree with contraction and tensor bond compression steps. Within this framework, sketched in Fig. 1, the problem reduces to optimizing a cost function over such computational trees: we term the macro-optimization over trees "hyperoptimization." As we will demonstrate in several examples, optimizing a simple cost function related to the memory or computational cost of the contraction also leads to an approximate contraction tree with small contraction error. Consequently, our hyperoptimized approximate contraction enables the efficient and accurate simulation of a wide range of graphs encountered in different tasks, bringing the possibility of eliminating, or otherwise improving on, formal exponential costs. In addition, in the structured lattices arising in many-body physics simulations, we observe that we can improve on the best physically motivated approximate contraction schemes in the literature.

A tensor T is a multi-index quantity (i.e., a multidimensional array). We use lower indices to index into the tensor, e.g., $T_{i_1 i_2 \ldots i_n}$ is an element of an n-index T, and upper indices to label a specific tensor, e.g., $T^{[1]}, T^{[2]} \ldots$, out of a set of tensors. A tensor network contraction sums (contracts) over the (possibly shared) indices of a product of tensors,

$$T_{\{e_{\text{out}}\}} = \sum_{\{e\}/\{e_{\text{out}}\}} \prod_v T^{[v]}_{\{e_v\}}, \tag{1}$$

where $\{e\}$ is the total set of indices, $\{e_{\text{out}}\}$ is the subset that is left uncontracted, and $\{e_v\}$ is the subset of $\{e\}$ for tensor $T^{[v]}$. We can place the tensors at the vertices v of a network (graph), with the bonds (edges) corresponding to

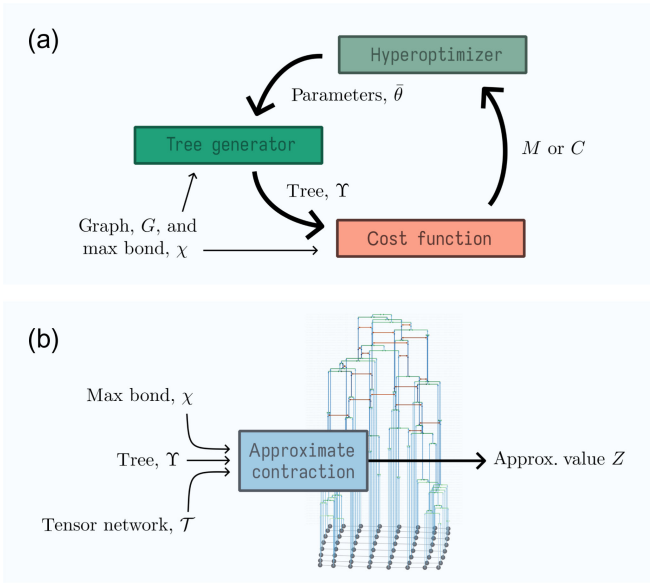

FIG. 1. Overview. The approximate contraction is specified by a sequence of contractions and compressions, expressed as an ordered tree. The strategy optimizes a cost function over such trees. (a) The hyperoptimization loop. Approximate contraction trees Υ on the graph G are suggested by the tree generator. The tree characteristics are controlled by heuristic parameters θ and maximum bond dimension χ. The hyperoptimizer minimizes a cost function M or C (peak memory or computational cost). (b) The approximate contraction tree. The tensor network $\mathcal{T}$ is shown at the bottom. Moving upward, pairs of tensors are contracted (blue lines), and singular value compressions are performed between tensors (orange lines). By the top of the tree, one obtains a scalar output Z, using resources $\sim M$ or C.

the indices $\{e\}$. An examples of contraction is shown in Fig. 2(a).

In practice, the sum in Eq. (1) is performed as sequence of pairwise contractions, and the order of contraction greatly affects both the memory and time costs. Much recent work has been devoted to optimizing contraction paths in the context of simulating quantum circuits [6–9]. Parametrized heuristics that efficiently sample the space of contraction paths, for example, by graph partitioning, are crucial, and optimizing the parameters of such heuristics (hyperoptimization) to minimize the overall cost has proven particularly powerful, leading to dramatic reductions in contraction cost (i.e., many orders of magnitude).

Here we extend the ideas of hyperoptimized tensor network contraction to the setting of approximate tensor network contraction. As discussed above, approximate contraction has a long history in many-body simulation, but such work has focused on regular lattices. Although several recent contributions have addressed arbitrary graphs [30,51,52], with a fixed contraction strategy, they do not focus on optimizing the strategy itself. In part, this is because there is a great deal of flexibility (and thus many components to optimize) when formulating an approximate contraction algorithm, and because an easily computable metric of quality is not clear *a priori*.

We proceed by first formulating the search space of approximate tensor network contraction algorithms, which we identify as a search over approximate contraction trees. To reduce the search space, we define simple procedures for gauging and when to compress bonds in the tree. We then discuss how to sample the large space of trees, by optimizing

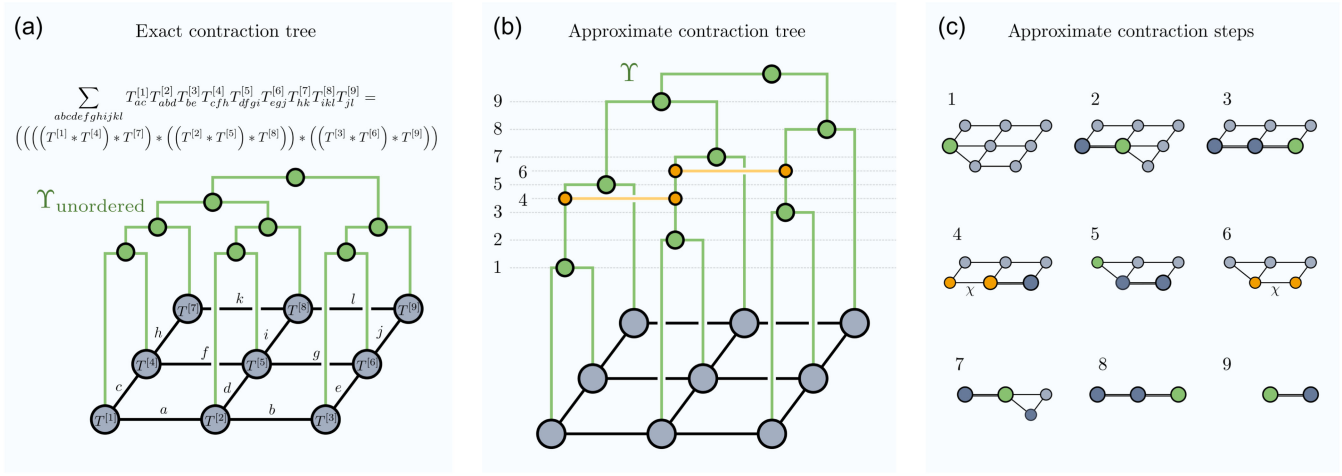

FIG. 2. (a) Pairwise exact contraction of a tensor network, with the unordered contraction tree $\Upsilon_{\text{unordered}}$ indicating the contractions. Each intermediate (green node) corresponds to a pair of parentheses in the expression. (b) An approximate contraction tree Υ for same network. Since compression steps do not commute, this tree is ordered. Here, the compressions (orange lines) take place at steps 4 and 6. (c) The sequence of contractions and compressions associated with the tree in (b). Newly contracted tensors in green, tensors with compressed bonds in orange.

the hyperparameters of a contraction tree generator, with respect to the peak memory or computational cost. We use numerical experiments to establish the success of the strategy, comparing to existing algorithms designed for structured lattices and for arbitrary graphs. Finally, using the hyperoptimized approximate contraction algorithms, we showcase the range of computational problems that can be addressed, in many-body physics, computer science, and complexity theory, illustrating the power of approximate tensor network computation.

II. FRAMEWORK FOR APPROXIMATE CONTRACTION ALGORITHMS

A. Components of approximate contraction

In an exact tensor network contraction, the computational graph, specified by the sequence of pairs of tensors which are contracted, can be illustrated as a computational contraction tree. This is illustrated in Fig. 2(a), where the tensor network is shown by the black lattice at the bottom, and the contractions between pairs occur at the green dots in tree, $\Upsilon_{\text{unordered}}$. Note that the value and cost of the exact tensor network contraction do not depend on the order in which the contractions are performed [53]; thus, the contraction tree is unordered. The problem of optimizing the cost of exact contraction is thus a search over contraction trees to optimize the floating point and/or memory costs.

In the process of contracting tensors, one generally creates larger tensors, which share more bonds with their neighbors. In approximate contraction we aim to reduce the cost of exact contraction by introducing an approximation error. The most commonly employed approximation is to compress the large tensors into smaller tensors (with fewer or smaller indices); this is the type of numerical approximation that we also consider here. The simplest notion of compression arises in matrix contraction, e.g., given two $D \times D$ matrices A, B, the contraction $AB \approx \bar{A}\,\bar{B}$, where $\bar{A}$ is of dimension $D \times \chi$ and B is of dimension $\chi \times D$, and the approximation is an example of a low-rank matrix factorization. The singular value decomposition (SVD) is an optimal (with respect to the Frobenius norm) low-rank matrix factorization. Singular value decomposition is also at the heart of compressing tensor network bonds. For example, if we have two tensors connected by bonds [Fig. 3(a)], we can view the bonds as performing a matrix contraction [Fig. 3(b)], and use SVD to replace the connecting bonds by one of dimension χ [Fig. 3(d)]. In the general tensor network setting, however, things are more complicated, because when compressing a contraction between two tensors, one should consider the other tensors in the network, which affect the approximation error. The effect of the surrounding tensors on the compression of a given bond is commonly known as including the "environment" or "gauge" into the compression. We consider how to perform bond compression, including a simple way to include environment effects into the bond compression for general graphs, in Sec. II C.

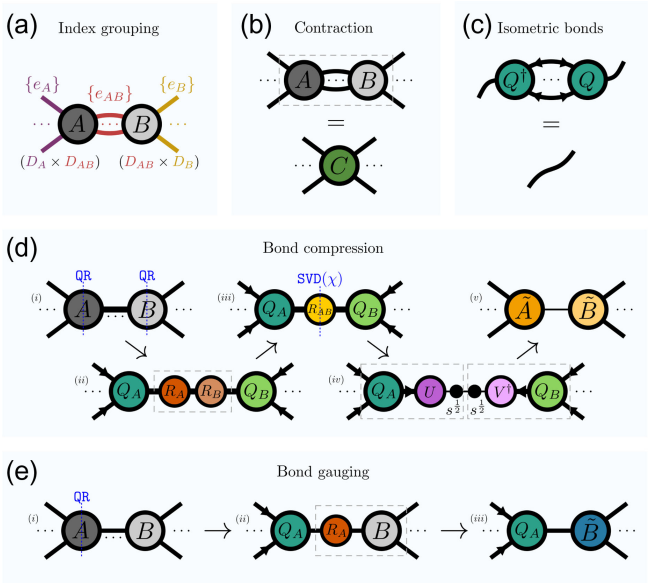

FIG. 3. Primitive tensor operators for approximate contraction. (a) Grouping of indices into left, shared, and right sets, giving a matricization of the product AB, with rank D_{AB}. (b) Graphical depiction of contracting two tensors $AB \to C$. (c) Graphical depiction of an isometric tensor Q such that when dimensions with incoming arrows are grouped $Q^\dagger Q = 1$. (d) Compression of the shared bonds between two tensors A and B, via QR reduction and truncated SVD to new shared bond dimension χ. (e) Gauging of the bond between A and B to generate an isometric tensor Q_A.

Given a compression method, we view the approximate tensor network contraction as composed of a sequence of contraction and compression steps. Compressions do not commute with contractions (or each other); thus, a contraction tree with compression (an approximate contraction tree) is an ordered tree. An example tree Υ is shown in Fig. 2(b), where in addition to the contraction operations (the green dots), we see compressions of bonds between tensors (the orange lines). The ordered sequence of contractions and compressions is visualized in Fig. 2(c). If we work in the setting where the compressed bond dimension χ is specified at the start, then once the approximate contraction tree is written down, the memory or computational cost of the contractions and compressions can be computed. Optimizing the approximate contraction for such costs thus corresponds to optimizing over the space of approximate contraction trees.

The space of trees to optimize over is extremely large. This we tackle in two ways: by defining the position of compression steps in the tree entirely in terms of where the contractions take place (discussed in Sec. II D), which means we only need to optimize over the order of the contractions; and by using the hyperoptimization strategy, where (families) of trees are parametrized by a small set of heuristic parameters, constituting a reduced dimensionality search space (described in Secs. II G and II H).

Ideally we wish to minimize the error of the approximate contraction as well as the cost, but the error is not known *a priori*. This can only be examined by benchmarking the errors of our hyperoptimized contraction trees. This is the subject of Sec. III.

Note that other ingredients could also be included in an approximate tensor network contraction algorithm, for example, the use of factorization to rewire a tensor network, generating a graph with more vertices [20,52], or inserting unitary disentanglers to reduce the compression error [54,55]. We do not currently consider these ingredients in our algorithm search space, although the framework is sufficiently flexible to include such ingredients in the future. We also note that some of these additional ingredients are targeted at the renormalization of loop correlations in tensor networks to yield a proper renormalization group (RG) flow [56]. We discuss the corner double line model and show that it is accurately contractible, at least at small bond dimension, with our approximate strategy in the Supplemental Material (SM) [57].

To aid in our discussion of the ingredients of the approximation contraction algorithm, and how to examine our choices, we will use a set of standard benchmark models, which we now discuss.

B. Models for testing

To assess our algorithmic choices, we will consider two families of lattices and two tensor models. (Note that these are only the tensor networks we use for testing the algorithm; Sec. IV further considers other models to demonstrate the power of the final protocol). The two types of lattices we consider are (i) the 2D square and 3D cubic lattices, which reflect the structured lattices commonly found in many-body physics applications, and (ii) 3-regular random graphs (graphs with random connections between vertices, where each vertex has degree 3). On these lattices, the two types of tensors we consider are (i) (uniaxial) Ising model tensors, at inverse temperature β close to the critical point, and (ii) tensors with random entries drawn uniformly from the distribution $[\lambda, 1]$ (we refer to this as the URand model). Changing λ allows us to tune between positive tensor network contractions and tensors with random signs, the latter case being reminiscent of some random circuit tensor networks. In all models, the dimension of the tensor indices of the initial tensor network will be denoted D, while the dimension of compressed bonds will be denoted χ; we refer to the value of the tensor network contraction as Z, and the free energy per site $f = -\ln Z/\beta N$, where N is the number of spins. More discussion of these models (as well as a treatment of corner double line models [56]) is in the SM [57].

C. Bond compression strategies

We first define how to compress the shared bonds $\{e_{AB}\}$ between tensors A, B. We can matricize these by grouping the indices as $\{e_A\}/\{e_{AB}\}$, $\{e_{AB}\}$, and $\{e_B\}/\{e_{AB}\}$, with effective dimensions D_A, D_{AB}, and D_B, respectively [see Fig. 3(a)]. Generally, $D_{AB} < D_A$ and $D_{AB} < D_B$ and so AB is already low rank and we can avoid forming it fully. Instead we perform QR-decomposition (QR) of the matricized A, B, giving

$$AB = Q_A(R_A R_B)Q_B \tag{2}$$

$$= Q_A(R_{AB})Q_B, \tag{3}$$

where the Q matrices satisfy the canonical conditions $Q_A^\dagger Q_A = 1$, $Q_B Q_B^\dagger = 1$, with the canonical direction indicated by an arrow in graphical notation shown in Fig. 3(c) (detailed in the SM [57]). Then, we obtain the compressed $\bar{A}$, $\bar{B}$ through the SVD of R_{AB},

$$\begin{aligned} R_{AB} &\approx U_A \sigma V_B^\dagger, \\ \bar{A} &= Q_A U_A \sigma^{1/2}, \\ \bar{B} &= \sigma^{1/2} V_B^\dagger Q_B, \end{aligned} \tag{4}$$

truncating to χ maximal singular values in σ. Because of the canonical nature of the Q matrices, truncating the SVD of R_{AB} achieves an optimal compression in the matrix Frobenius norm of AB due to the orthogonality of Q_A, Q_B.

Usually $\mathcal{T}$ will contain additional tensors connected to A, B. We refer to the additional network of connected tensors as the environment E, with $\mathcal{T} = \sum_{\{e\}/\{e_{\text{out}}\}} A_{\{e_A\}} B_{\{e_B\}} E_{\{e_E\}}$ [Fig. 4(a)]. To compress the bond e_{AB} optimally, we must account for E. We first consider the case when E forms a tree around the bond e_{AB} [Fig. 4(a)(iii)]. Then, we can perform QR inward from the leaves of the tree, pushing the R factors toward the bond [Figs. 4(b)(i)–4(b)(iii)]. This is a type of gauging of the tensor network (i.e., it changes the tensors but does not change the contraction $\mathcal{T}$), and we refer to this as setting the bond e_{AB} in the tree gauge; alternatively, we can say the tensors in the tree are in the canonical form centered around bond e_{AB}. This results in a similar matricized $\mathcal{T} = Q_A(R_{AB})Q_B$, where Q_A, Q_B have accumulated the products of R factors from all tensors to the left and right of bond e_{AB} [Fig. 4(b)(iii)]. Then, the truncated SVD of R_{AB} in Eq. (4) similarly achieves an optimal compression of e_{AB} with respect to error in $\mathcal{T}$.

More generally, $\mathcal{T}$ may contain loops, which extend into the environment [Fig. 4(a)] and a similarly optimal gauge is hard to compute [58,59]. However, by cutting loops in the environment E (i.e., not contracting some of the bonds in the loops) we obtain a tree of tensors around bond e_{AB}, e.g., a spanning tree out to a given distance r. (There are multiple ways to cut bonds to obtain a spanning tree; the specific spanning tree construction heuristic is given in the SM [57].) Placing e_{AB} in the tree gauge (of distance r), we can then perform the same compression by truncated SVD, but without the guarantee of optimality since we are

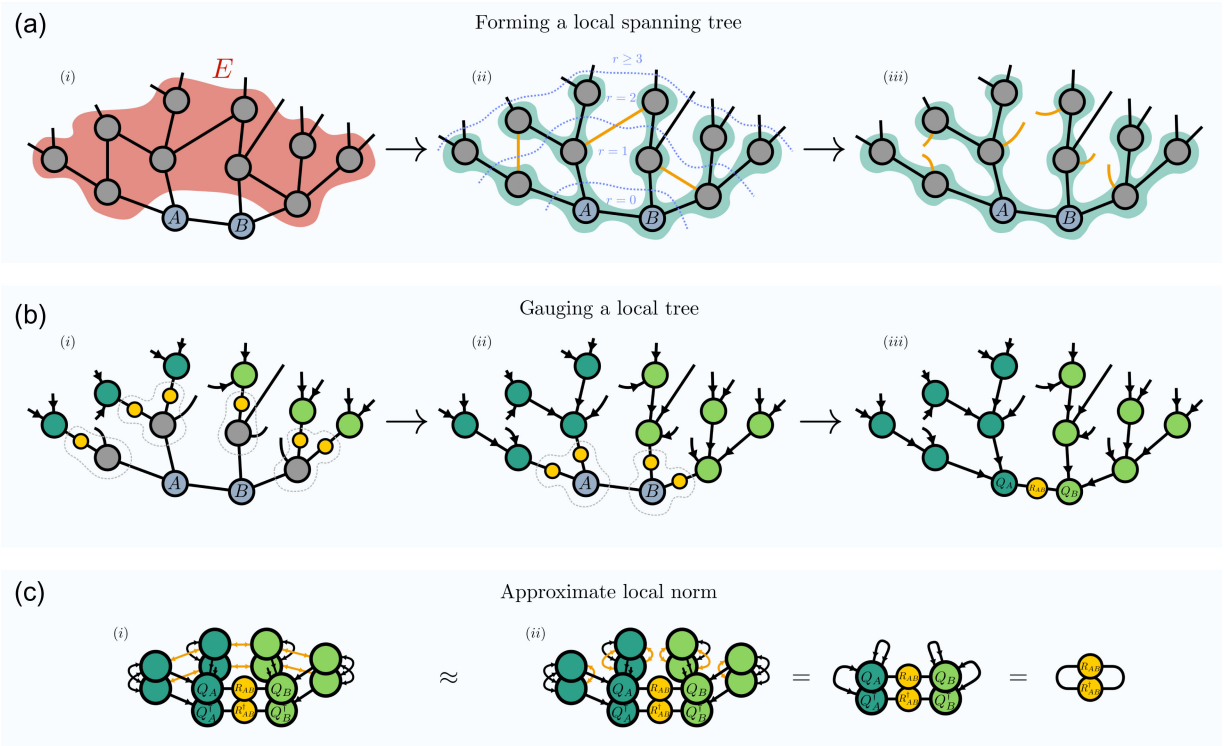

FIG. 4. Overview of the tree gauge for improving bond compression accuracy, suitable for arbitrary local geometry. (a) Given the bond e_{AB} connecting tensors A and B, we want to take into account information from the surrounding environment E, shown in (i). In (ii) we form a local spanning tree (shaded bonds) up to distance $r = 2$ from A and B. If we consider "loop" bonds (colored orange) that are not part of the spanning tree as cut, then the resulting local tree environment shown in (iii) can be optimally compressed as a proxy target. (b) Depiction of the gauging process for a local tree. In (i) tensors at distance $r = 2$ from A and B are QR decomposed, and the R factors (yellow circles) are accumulated toward the bond e_{AB}; see Fig. 3(e). In (ii) the same happens for $r = 1$ tensors, and finally in (iii) the R factors from A and B after accumulating all the outer gauges, R_A and R_B, are contracted to form R_{AB}. (c) The Frobenius norm (squared) of an $r = 1$ local region in the tree gauge is shown in (i). The norm of the local network with loop bonds (orange) cut is shown in (ii), which is exactly encoded, due to the isometric tensors, as $(\mathrm{Tr}R_{AB}^{\dagger}R_{AB})^{1/2}$. Performing a truncated SVD on R_{AB} is thus only r-locally optimal up to the presence of such loop bonds.

neglecting loop correlations; see Fig. 4(c). However, this type of tree gauge compression is easy to use in the general graph setting, and thus will be the main bond compression scheme explored in this work.

One can show [57,60–62] that performing the truncation in Eq. (4) is equivalent to inserting the projectors $P_A = R_B V_B \sigma^{-1/2}$ and $P_B = \sigma^{-1/2} U_A^{\dagger} R_A$ such that $AB \approx AP_AP_BB$. As such, having computed R_A and R_B from the local spanning trees we can form and contract P_A and P_B directly into our original tensor network without affecting any tensors other than A and B, but still include information from distance r away. In other words, the steps depicted in Fig. 4 are performed virtually, which avoids having to reset the gauge after compression.

D. Early versus late compression

In practice, compression must be performed many times during a tensor network contraction. It might seem natural to perform compression immediately after two tensors are contracted to form a tensor larger than some size threshold, here given by a maximum bond dimension χ (early compression). This is illustrated in Fig. 5(a). However, as discussed above, including information from the environment is important for the quality of compression. Early compression means that tensors in the environment are already compressed, decreasing their quality. An alternative strategy is to compress a bond between tensors only when one of them (exceeding the size threshold) is to be contracted (late compression), as illustrated in Fig. 5(b). By delaying the compression, more bonds or tensors in the environment are left uncompressed, which can potentially improve the quality of the contraction. However, late compression will also increase the cost or memory of contraction (as there are more large tensors to consider). This means that it is most efficient to use late compression when the associated gain in accuracy is large.

In Fig. 5(c), we assess the effect of early versus late compression when contracting a 2D 16×16 lattice ($D = 4$, URand model with tensor entries $\in [-0.5, 1]$). All compressions are performed using the tree gauge (out to some distance r, several tree distances r are shown), and we show

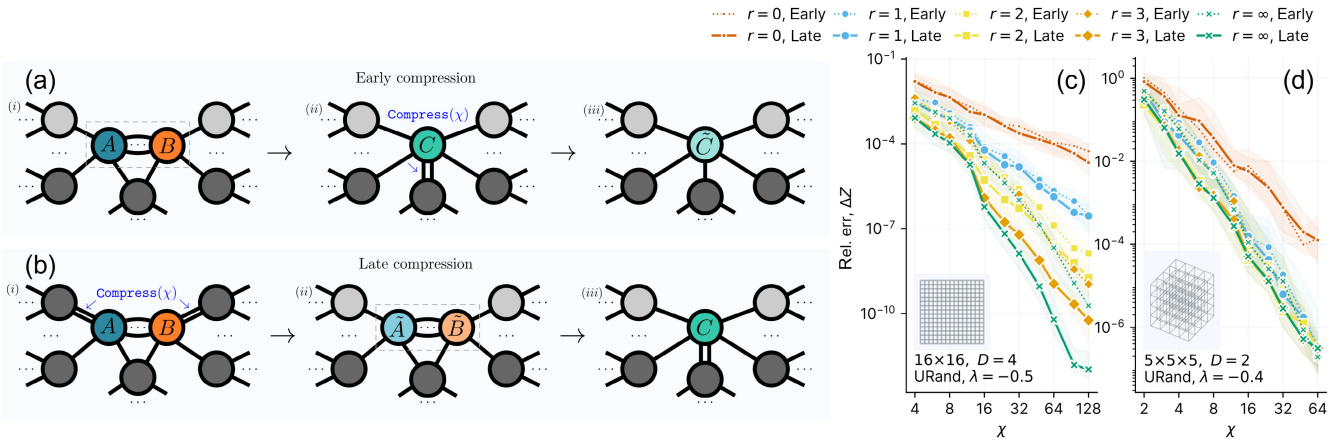

FIG. 5. (a) Schematic of "early" compression, where *after* each pairwise contraction, any shared bonds of total size $> \chi$ are truncated. (b) Schematic of "late" compression, where *before* each pairwise contraction, any shared bonds of total size $> \chi$ are truncated. (c) Error ΔZ of contracting a 16×16 $D = 4$ TN with uniform random entries $\in [-0.5, 1]$ as a function of χ, tree gauging distance r, and either early or late compression. The TN is contracted using the standard MPS boundary contraction algorithm. Line (band) shows median (interquartile range) over 50 instances. (d) The same but for an approximate contraction of a $5 \times 5 \times 5$ $D = 2$ tensor network with uniform random entries $\in [-0.4, 1]$. The 3D TN is contracted using a hyperoptimized `Span` tree.

the relative error of the contraction ΔZ as a function of the maximum allowed bond dimension χ. We see in this case that late compression is more accurate than early compression, and that this improvement increases when using larger tree gauge distances, reflecting the fact that the gauging is incorporating more environment information. In Fig. 5(d), we similarly compare early versus late compression using the tree gauge in a 3D lattice (using a hyperoptimized `Span` tree as described later). In contrast to the 2D result, here we see a smaller improvement from performing late versus early compression and from increasing the tree gauge distance. This suggests that incorporating the effect of the environment requires a more sophisticated gauging strategy in 3D. In general, we summarize our findings as follows: late compression is preferred when trying to maximize accuracy for a given bond dimension χ or size of the largest single tensor operation, while early compression can be better when optimizing computational total cost or memory for a given accuracy. In our subsequent calculations, we will indicate the choice of early or late compression in the simulations.

E. Comparison of the tree gauge to other gauges

To evaluate the quality of the tree gauge compared to other gauging or environment treatments in the literature, we consider contractions on a 2D lattice. To isolate the comparison to only the choice of gauge, we use the same approximate contraction tree as used in boundary contraction; namely, contraction occurs row by row starting from the bottom, and compression occurs left to right after the entire row is contracted. We then use four different gauges or environment treatments during the compression: none, tree, boundary, and full. None corresponds to no gauging. Tree is the tree gauge discussed above (up to distance r). Boundary corresponds to the standard matrix product state (MPS) boundary gauging [44,50], where, after the new row of tensors has been contracted into the boundary, the boundary MPS is canonicalized around the leftmost tensor and then compressed left to right in an MPS compression sweep (see Fig. 3 of the SM for an illustration [57]). Full corresponds to explicitly computing the environment E by approximate contraction (using the standard MPS boundary contraction algorithm to contract rows from the top). Then, for the tensors A, B sharing bond e_{AB} to be compressed, the scalar value of the tensor network is $Z = \mathrm{Tr}BEA$ (where A, B, E have been matricized). Using the eigenvector decomposition, $BEA = L\sigma R^\dagger$, where L, R are left, right eigenvectors, respectively, then e_{AB} is optimally compressed by defining $\tilde{B} = \tilde{L}^\dagger B$, $\tilde{A} = A\tilde{R}$, where $\tilde{L}$, $\tilde{R}$ are the eigenvectors corresponding to the eigenvalues of largest absolute magnitude [22,24]. Note that the full environment gauge is expensive, as it requires an estimate of E from all the tensors in the network.

The numerical performance of the different strategies is shown in Fig. 6 for two problems: a 32×32 lattice (2D Ising model, near critical) and a 16×16 lattice ($D = 4$, URand model with entries $\in [-0.5, 1]$). In all cases, we see that including some environment information is better than not including any environment ("none"). In the 2D Ising model, as the tree distance r increases, tree gauge compression converges in quality to the MPS boundary environment scheme ("boundary"); the two are related as the MPS boundary corresponds to setting an infinite tree distance r for a tree that grows only along the boundary. In the 2D URand model, even for small r, the tree gauge already improves on the boundary environment. The full environment treatment yields the best compression quality for larger χ, but this is achieved at larger cost.

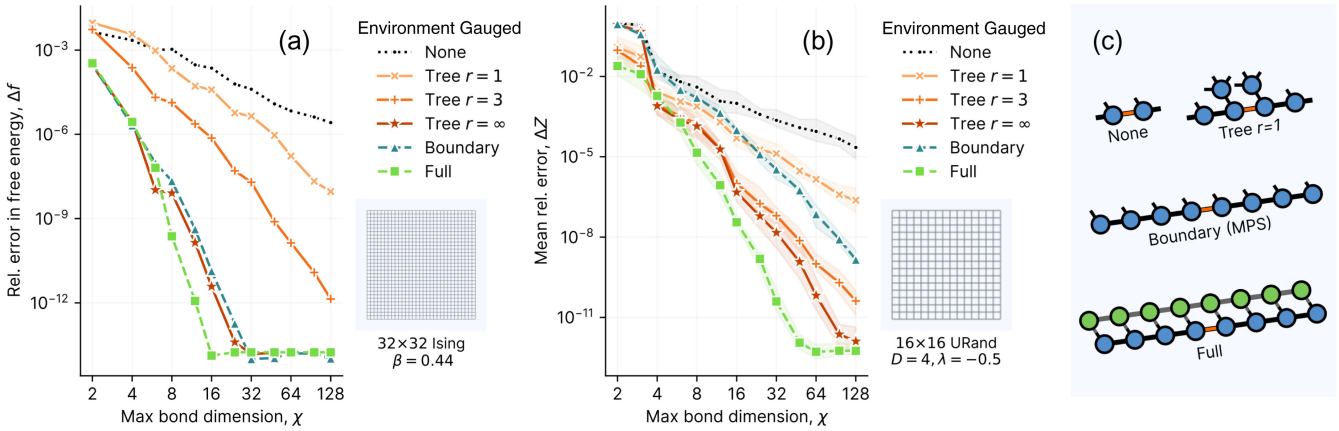

FIG. 6. (a) Error in free energy Δf as a function of bond dimension χ for different gauging and environments for the 2D Ising model at the critical point. (b) Contraction error ΔZ for the same settings but on a $D = 4$ URand model with $\lambda = -0.5$. All contractions contract from the boundary row by row; thus all bond compressions are for bonds on the boundary. However, different gauging is performed before the compressions. None, no gauging; tree, bonds are placed in the tree gauge up to distance r, followed by "late" compression, Boundary, bonds are placed in the canonical form of the MPS boundary, before compression; full, the environment around tensors A, B is explicitly contracted using a counterpropagating MPS of the same bond dimension, and the bond between AB is then truncated to minimize the error in TrBEA. Note that since E is itself only approximate and many truncations are compounded, the error overall is not guaranteed to be smaller than another method—as seen for some small χ points here. (c) Illustration of the different environments that a single compression step is optimal with regard to.

Our numerical results in 2D suggest that the tree gauge is a reasonable compromise between accuracy and efficiency, equaling or outperforming the common boundary environment strategy, while being well defined for more general graphs.

F. Approximate contraction algorithm

Given a choice of late or early compression, and using the tree gauge, we can explicitly write down a simple pseudo-code version of the core approximate contraction function, Algorithm 1, which implements Fig. 1(b). The exact form of the inner functions is detailed in the SM [57]. An alternative, that might be useful in some contexts, is to use the compression locations to transform a tensor network into an approximately equivalent but exactly contractible form, by inserting a set of explicit projectors—this is also detailed in the SM [57].

G. Generating contraction trees

After fixing the choice of early or late compression, the subsequent location of compressions in the contraction tree is purely determined by the contraction order. This is a major simplification, because, when optimizing over the approximate contraction trees, we need only optimize the order of contractions. Nonetheless, the space of ordered trees is still extremely large and hard to sample fully.

To simplify the search, we work within a lower-dimensional parametrization of the search space by introducing tree generators. These heuristics generate trees within three structural families we term `Greedy`, `Span`, and `Agglom`. The specific instance of tree within each family is defined by a set of hyperparameters that can then be optimized. Here we describe the heuristic generators at a high level (with a more detailed description in the SM [57]). The input to the generators is only the tensor network

ALGORITHM 1. Approximate contraction.

Input: tensor network $\mathcal{T}$, ordered contraction tree Υ, maximum bond dimension χ, tree gauge distance r, flag `compress_late`
// i, j, k, l label tensors, $T^{[i]}$, … in $\mathcal{T}$.
for $i, j \in \Upsilon$ **do**
  if `compress_late` **then**
    for $l \in$ NEIGHBORS$(\mathcal{T}, i)$ **do**
      if BONDSIZE$(\mathcal{T}, i, l) > \chi$ **and** $l \neq j$ **then**
        COMPRESS_TRUE_GAUGE$(\chi, r, \mathcal{T}, i, l)$
      end if
    end for
    for $l \in$ NEIGHBORS$(\mathcal{T}, j)$ **do**
      if BONDSIZE$(\mathcal{T}, j, l) > \chi$ **and** $l \neq i$ **then**
        COMPRESS_TRUE_GAUGE$(\chi, r, \mathcal{T}, j, l)$
      end if
    end for
  end if
  $k \leftarrow$ CONTRACT$(\mathcal{T}, i, j)$
  if not `compress_late` **then**
    for $l \in$ NEIGHBORS$(\mathcal{T}, k)$ **do**
      if BONDSIZE$(\mathcal{T}, k, l) > \chi$ **then**
        COMPRESS_TRUE_GAUGE$(\chi, r, \mathcal{T}, k, l)$
      end if
    end for
  end if
end for
Return: $\mathcal{T}$

graph, bond sizes, and χ—the tensor entries are not considered.

The `Greedy` tree generator assigns a score to each bond in the $\mathcal{T}$. It then chooses the highest scoring bond, generates a new $\mathcal{T}$ by simulating bond contraction and compression (i.e., computing the new sizes and network structure), and repeats the process, building an ordered tree. The bond score is a combination of the tensor sizes before and after (simulated) compression and contraction, a measure of the centrality of the tensors (their average distance to every other tensor), and the subgraph size of each intermediate (i.e., how many tensors were contracted to make the current tensor); the hyperparameters are the linear weights of each component in the score.

The `Span` tree generator is inspired by boundary contraction. It generates a directed spanning tree of the original graph, and the contraction order is chosen to contract the leaves inward. Contracting simultaneously along all the branches of the tree defines an effective contraction boundary, that sweeps toward the root. The algorithm begins by choosing either the most or least central node in the tensor network (TN), i_0, as an initial span, $\mathcal{S} = \{i_0\}$. It then greedily expands to a connected node j, adding the contraction $(i, j) \to i$ in *reverse* order to the path, where $i \in \mathcal{S}, j \notin \mathcal{S}$. The algorithm repeats by then considering all neighbors of the newly expanded span $\mathcal{S} \to \mathcal{S} \cup \{j\}$. A few local quantities—connectivity to $\mathcal{S}$, dimensionality, centrality, and distance to i_0—are combined into a score used in the greedy selection of the next node in the tree, and the combination weights are the hyperparameters.

The previous approaches grow ordered trees locally. The `Agglom` tree generator explicitly considers the full TN from the start and is inspired by renormalization group contraction strategies in the literature [20]. Given a community size K, the generator performs a balanced partitioning over the N tensors in $\mathcal{T}$ to find N/K roughly equal subgraphs. These subgraphs then define intermediate tensors, and the tensors within the subgraph are contracted using the `Greedy` algorithm with default parameters. After simulating the sequence of compressions and contractions, the network of intermediate tensors defines a new "coarse grained" tensor network for which the agglomerative process can be repeated. In this work, `Agglom` uses the KaHyPar graph partitioner [63,64], treating the community size K, imbalance, partitioning mode, and objective as the tunable hyperparameters.

Some sample ordered contraction trees generated by the above heuristics are shown in Fig. 7 for a 2D 8×8 lattice. In particular, we observe the boundarylike contraction order of the `Span` tree (contracting row by row from the bottom) and the hierarchical RG-like structure of the `Agglom` tree (forming increasing clusters); the `Greedy` tree contracts simultaneously from all four corners inward, rather than from one side like the `Span` tree. Note that the `Agglom` tree tends to perform more contractions before compressions are performed than the `Span` tree because it constructs many separate clusters simultaneously, and the `Greedy` tree exhibits behavior intermediate between the two.

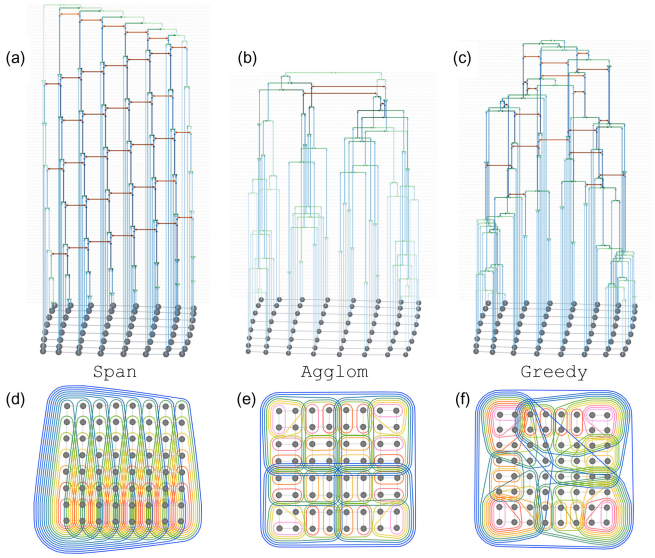

FIG. 7. Example approximate contraction trees for a 2D 8×8 square lattice TN. Panels (a)–(c) show the ordered contraction trees with compressions (orange, using the "late" strategy) explicitly marked for the `Span`, `Agglom`, and `Greedy` methods, respectively. The computation proceeds from the bottom to top. Panels (d)–(f) show the same contraction trees as hierarchical communities, with the contraction ordering proceeding from the smallest pink bands to largest blue bands.

H. Optimizing the contraction trees

We optimize the trees by tuning the hyperparameters that generate them with respect to a cost function. Since we also wish to sample many different trees it is important that the cost function is cheap to evaluate. We perform the optimization over the hyperparameter space using Bayesian optimization [65,66], which is designed for gradient-free high-dimensional optimization. The overall process is shown in Fig. 1(a), with more detailed pseudo-code in the SM [57].

Depending on the computational resources available, we can choose the cost function to be memory (peak memory usage M) or the computational (floating point) cost C. For C we include the cost of contractions, QR, and SVD decompositions.

We optimize the contraction trees over the hyperparameters in each of the three families of ordered tree generators. In all results with optimized trees, we used a budget of 4096 trees, though in practice a few hundred often achieves the same result. The practical effect of the hyperoptimization time is considered in the SM [57].

I. Quality of hyperoptimization

To test the quality of the hyperoptimization and the tree search space, we first consider a small (but nonetheless

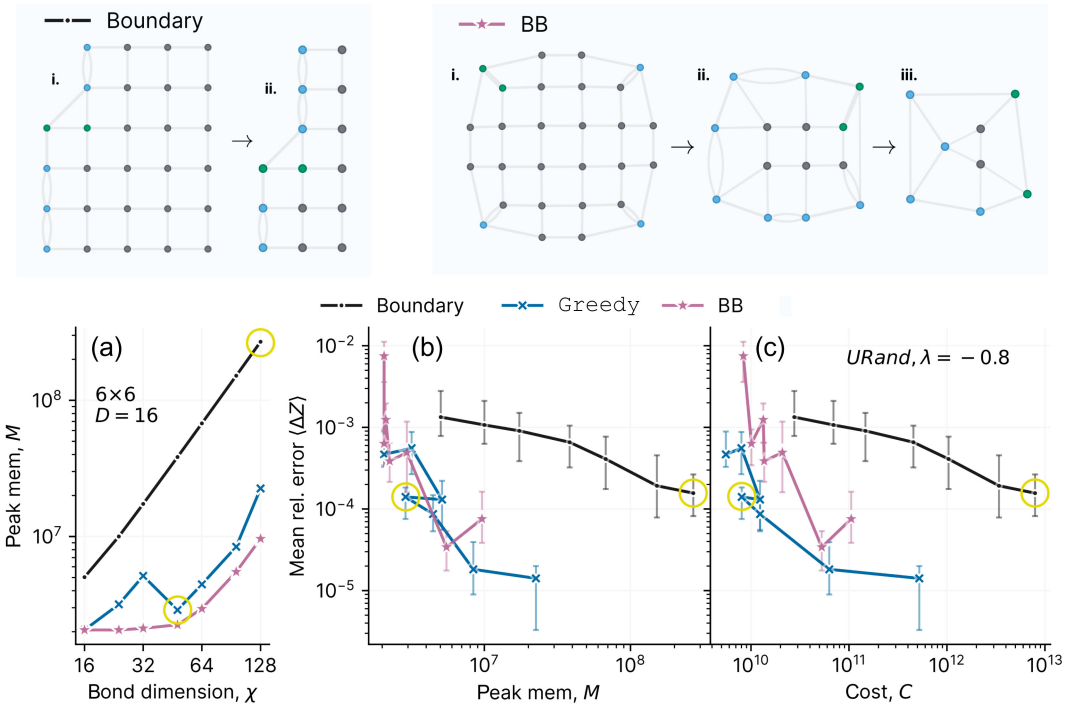

FIG. 8. Performance of hyperoptimized approximate contraction for a 6×6 square TN with all bonds of size $D = 16$. The TN is filled with uniformly random entries $\in [\lambda = -0.8, 1.0]$ (2D URand model). We test three different contraction trees: standard MPS boundary contraction, optimization over `Greedy` trees, brute force optimization over all approximate contraction trees (BB). The upper insets show snapshots illustrating the boundary contraction, an example of `Greedy`, and the optimal BB contraction for $\chi = 32$. (a) The peak memory usage M of the standard MPS boundary method, compared to the `Greedy` and BB algorithms which have been optimized for each χ. (b),(c) The error for each plotted against peak memory M and computational cost C, respectively, averaged over 20 instances of the URand model. The compressions are performed late using a tree-gauge distance $r = 1$. The yellow circles correspond to approximately equal error $\Delta Z \approx 10^{-4}$, using the boundary and `Greedy` trees, to enable the ratio of costs to be determined; e.g., the observed speed-up of `Greedy` over boundary for this error is 120×.

nontrivial-to-contract due to large D and χ) square TN of size 6×6. In this case, the number of tensors is small enough that it is possible to perform an exhaustive search over all ordered contraction trees using a branch and bound algorithm (BB; see SM [57]); here we minimize peak memory M. In Fig. 8 we compare the performance of the hyperoptimized `Greedy` algorithm against the exhaustive branch and bound search (using tree gauging for compression in each case for a fair comparison). The standard boundary contraction algorithm is also shown as a comparison point.

As shown in Fig. 8(a), hyperoptimizing over the space of `Greedy` trees produces performance quite similar to the BB search and very different from the standard boundary contraction. This indicates that the hyperoptimization is doing a good job of searching the approximation contraction tree space for graphs of this size. In the top panel, we can see the optimal contraction strategy found by BB produces a very different contraction order to boundary contraction, exploiting the finite size of the graph and the targeted χ to significantly reduce M.

We can also verify that optimizing M leads to reduced error. In Figs. 8(b) and 8(c), we show the contraction error for the URand model with $\lambda = -0.8$, where it can be seen that for equivalent error ($\sim 10^{-4}$, indicated by the yellow circles) the peak memory or cost of using the hyperoptimized `Greedy` or BB approximate contraction trees is indeed much lower than that of boundary contraction. Interestingly, the heavily optimized BB tree does not improve on the error of the `Greedy` tree for a given peak memory M.

III. BENCHMARKING HYPEROPTIMIZED APPROXIMATE CONTRACTION TREES

A. Summary of hand-coded strategies for regular lattices

In our benchmarking below, when considering regular lattices, we will compare to a range of hand-coded contraction strategies used in the literature in many-body physics applications, namely boundary contraction, corner transfer renormalization group (CTMRG) [67], and higher-order tensor renormalization group (HOTRG) [68]. We briefly summarize the hand-coded strategies here. Boundary contraction (as already used above) is a standard method in 2D, but has not been widely applied in 3D.

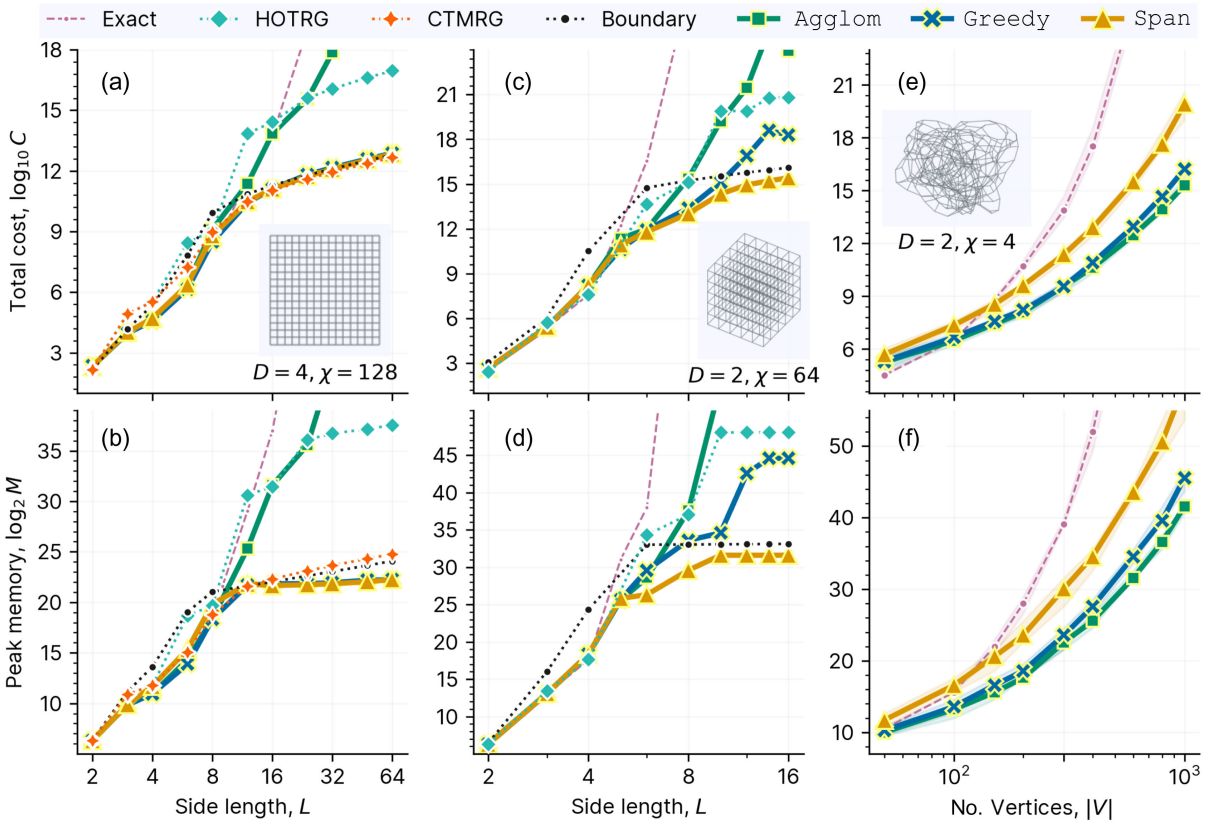

FIG. 9. Contraction peak memory and cost of approximate contraction trees in different families versus exact contraction cost and CTMRG, HOTRG, and boundary contraction algorithms, for different geometries (insets show sample geometry). The tree gauging distance is set to $r = 0$ for this comparison, with "early" compression, and we also turn off gauging in the boundary algorithm. The hyperoptimized trees are optimized for M and C separately. (a),(b) Peak memory M and cost C of contracting a 2D square TN with $D = 4$ and $\chi = 32$ as a function of side length L. (c),(d) Peak memory M and cost C of contracting a 3D cube TN with $D = 4$ and $\chi = 32$ as a function of side length L. (e),(f) Peak memory M and cost C of contracting 3-regular random graphs with initial bond dimension $D = 2$ and $\chi = 4$ as a function of number of vertices $|V|$. The line and shaded band show the median and interquartile range across 20 instances, respectively.

We define a 3D version of projected entangled pair state (PEPS) boundary contraction on a cube that first contracts from one face of the cube toward the other side, leaving a final 2D PEPS tensor network that is contracted by 2D boundary contraction (with the same χ). CTMRG is usually applied in 2D and to infinite systems. Here we apply CTMRG to the finite lattice by using a finite number of CTMRG moves [57]. Finally, HOTRG has been applied to both 2D and 3D infinite simulations; here we perform a limited number of RG steps appropriate for the finite lattice. For both CTMRG and HOTRG, we also compute and insert different projectors for each compression, since we are dealing with generically inhomogeneous systems. Illustrations of all the algorithms are given in the SM [57].

B. Cost scaling with graph size

In Fig. 9 we show the computational cost (memory and floating point cost) of hyperoptimized trees in the `Greedy`, `Span`, and `Agglom` classes for a 2D square of size $L \times L$, a 3D cube of size $L \times L \times L$, and for 3-regular random

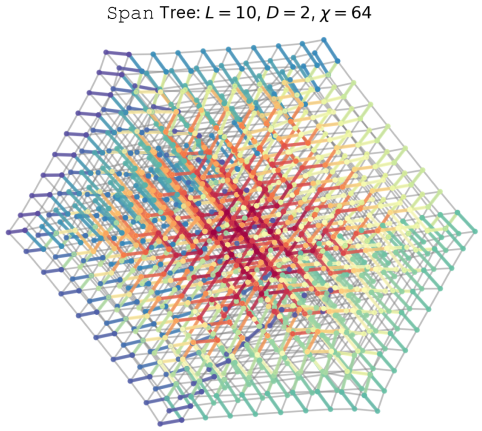

FIG. 10. Depiction of the contraction tree found by the `Span` algorithm, optimized for minimum floating point cost C, for a large 3D lattice. The colors of the highlighted edges and nodes indicate the stage of contraction they are involved in, running from blue (earliest) to red (latest).

graphs with $|V|$ vertices, using the early compression strategy, and given bond dimension χ. We compare against the cost of contraction trees generated by boundary contraction, CTMRG, and HOTRG.

From this and other examples, we can make some general observations. First, `Span` trees yield good costs for simple lattices which have a regular local structure, the `Agglom` tree is superior for random graphs, and the `Greedy` tree works well for both sets. The markedly different performance of the `Agglom` and `Span` trees on random versus simple lattices suggests that these are two good limiting cases for testing contraction heuristics. Interestingly, in the 3D cubic case, the hyperoptimized `Span` tree performs a boundarylike contraction, but rather than contracting from one face across to the other side, it can find a strategy that contracts all faces toward a point, as visualized in Fig. 10. This substantially improves over the hand-coded boundary PEPS strategy in terms of cost. Similar observations apply to the `Greedy` tree, which is similar to or outperforms both `Span` and hand-coded algorithms for smaller structured lattices, although its performance degrades for larger lattices. We also find that `Greedy` trees optimize the cost function less well in other instances of large lattices. This suggests that the search space generated by the `Greedy` tree generator is limiting at larger lattice sizes.

In the 2D square lattice, we find that compared to the hand-coded algorithms, the `Span` and `Greedy` trees with early compression are superior with respect to memory and cost, even beating out the most widely used boundary MPS strategy. At smaller system sizes, the superior performance of `Span` and `Greedy` over boundary MPS reflects the

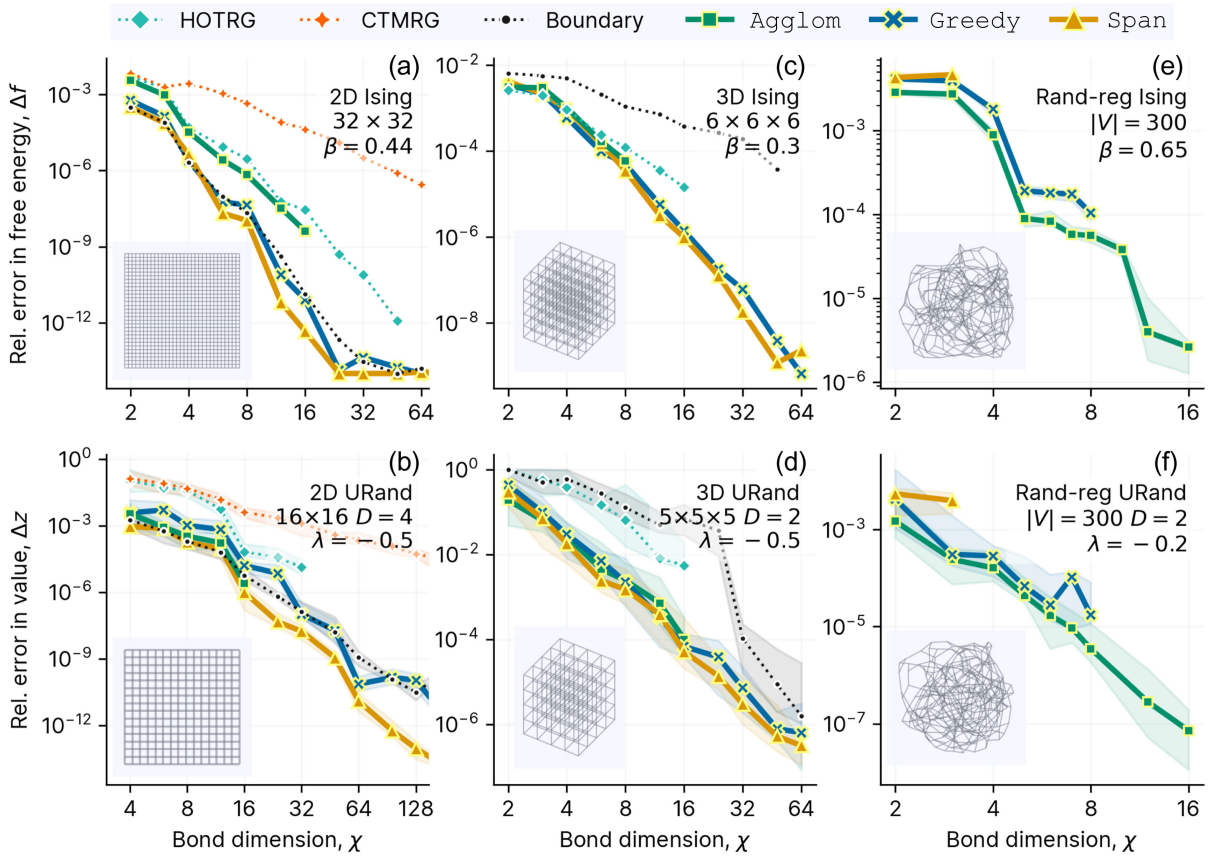

FIG. 11. Error versus compressed bond dimension χ of hyperoptimized approximate contraction using optimized `Span`, `Greedy`, and `Agglom` trees, in comparison to boundary contraction, CTMRG, and HOTRG, for medium size TNs. (Insets show sample geometry.) In terms of gauging settings for the hyperoptimized methods, in 2D we use $r = 6$ with late compressions, and for the others we use $r = 3$ and early compression. (a) Relative error in the free energy of the 2D Ising model on a 32×32 square lattice close to the critical point. (b) Relative error in the contracted value of the 2D URand model on a 16×16 square lattice with $D = 4$ in the intermediate hardness regime of λ. (c) Relative error in the free energy of the 3D Ising model on a $6 \times 6 \times 6$ cubic lattice close to the critical point. (d) Relative error in the contracted value of the 3D URand model on a $5 \times 5 \times 5$ cubic lattice with $D = 2$ in the intermediate hardness regime of λ. (e) Relative error in the free energy of the Ising model on 3-regular random graphs with $|V| = 300$ close to the critical point (line and bands show median and interquartile range across 20 instances). (f) Relative error in the contracted value of the URand model on 3-regular random graphs with $D = 2$ in the intermediate hardness regime of λ (line and bands show median and interquartile range across 20 instances).

ability of these algorithms to exploit edge and boundary effects. Interestingly, CTMRG is also superior to boundary MPS at small system sizes, and the optimized strategies seem to interpolate between a more CTMRG-like and MPS boundarylike contraction. The HOTRG algorithm exhibits similar performance to the `Agglom` tree, as expected due to its real-space RG motivated ordering of contractions. At larger sizes, boundary, CTMRG, `Span`, and `Greedy` show similar asymptotic cost, with the optimized strategies retaining a modest asymptotic improvement in memory.

C. Error versus bond dimension

As discussed above, we optimize over the generated contaction trees for a given bond dimension χ. In Fig. 11, we plot the relative error in the contraction value ΔZ or free energy per site Δf for 2D and 3D Ising and random tensor models for the hyperoptimized contraction and hand-coded strategies as a function of bond dimension.

It is natural to expect the error of an approximate contraction to decrease as we increase χ, since in the limit $\chi \to \infty$ the algorithm becomes exact. For all the models and algorithms investigated we find a roughly polynomial suppression of the error with inverse χ. What is perhaps less obvious is whether approximate contraction trees with given χ should yield comparable errors regardless of the cost of the particular tree, M or C. We see that this is in fact the case for the hyperoptimized trees, i.e., the error correlates reasonably well with the compressed bond dimension χ, independent of the choice of tree. Thus by choosing the optimized tree with lowest cost for a given χ, we are not paying a price in terms of accuracy.

On the other hand, the hand-coded algorithms do not follow this observation; e.g., CTMRG in the 2D lattice and boundary PEPS and HOTRG in the 3D lattice exhibit considerably larger errors than the hyperoptimized strategies for given χ. For fixed bond dimension, the hyperoptimized contraction trees appear to use the computational resources (memory and cost) in a more effective way to reduce error than the hand-coded strategies.

D. Error versus cost

We next consider the error obtained for a given peak memory or computational cost. In Fig. 12 we show the relative error in the contraction value ΔZ/free energy per site Δf for 2D and 3D Ising and random tensor models for the hyperoptimized contraction and hand-coded strategies, plotted against peak memory usage or contraction cost (depending on which was used as the cost function to optimize the trees). In this figure, the sizes of the problems were chosen so that the exact value of Z or f can be

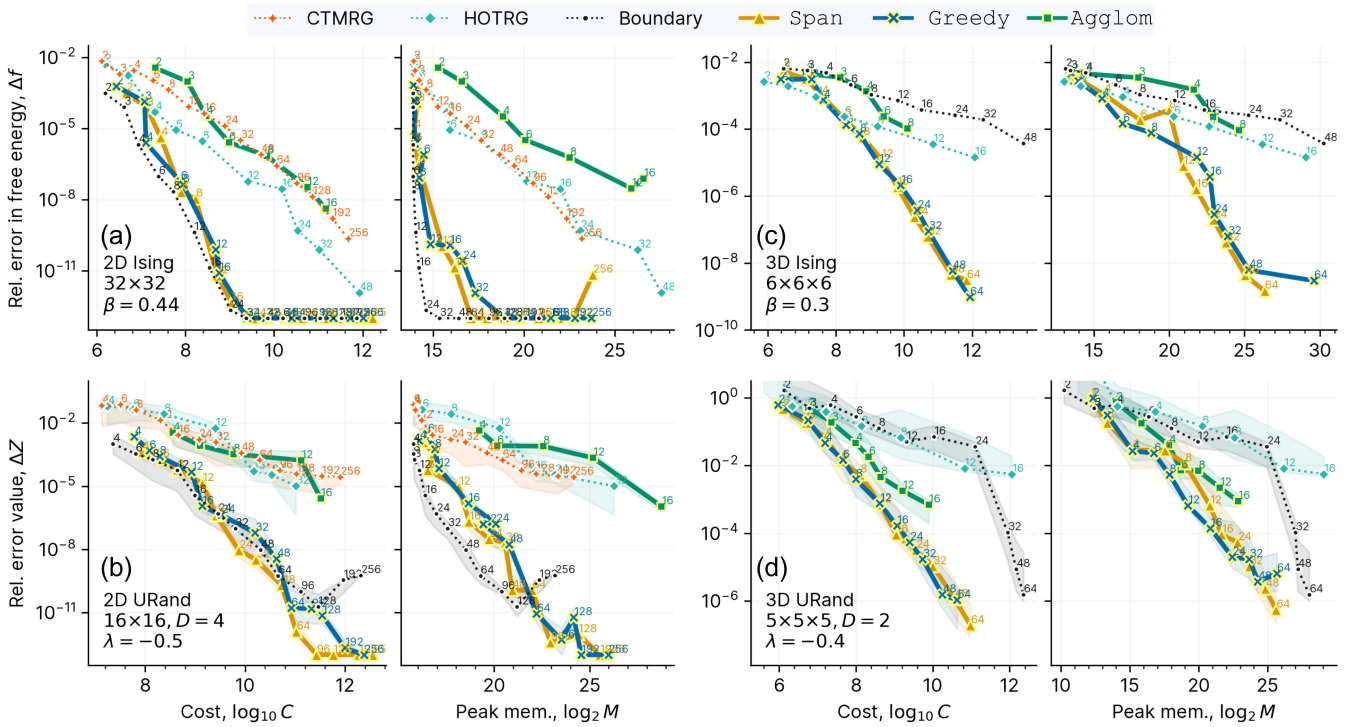

FIG. 12. Error versus cost of hyperoptimized approximate contraction using optimized `Span`, `Greedy`, and `Agglom` trees, in comparison to boundary contraction, CTMRG, and HOTRG, for medium size TNs where the exact reference values are available. Here the error is plotted against either the total cost of the contraction C or peak memory requirement M as computed by tracing through the computation. The trees are optimized for each separately. Four different lattice and model combinations are shown: (a) 2D Ising model, (b) 2D URand model, (c) 3D Ising model, and (d) 3D URand model. The lines are annotated with the value of χ. The gauging settings used for the hyperoptimized methods are $r = 6$ and late compression in 2D and $r = 2$ and early compression in 3D.

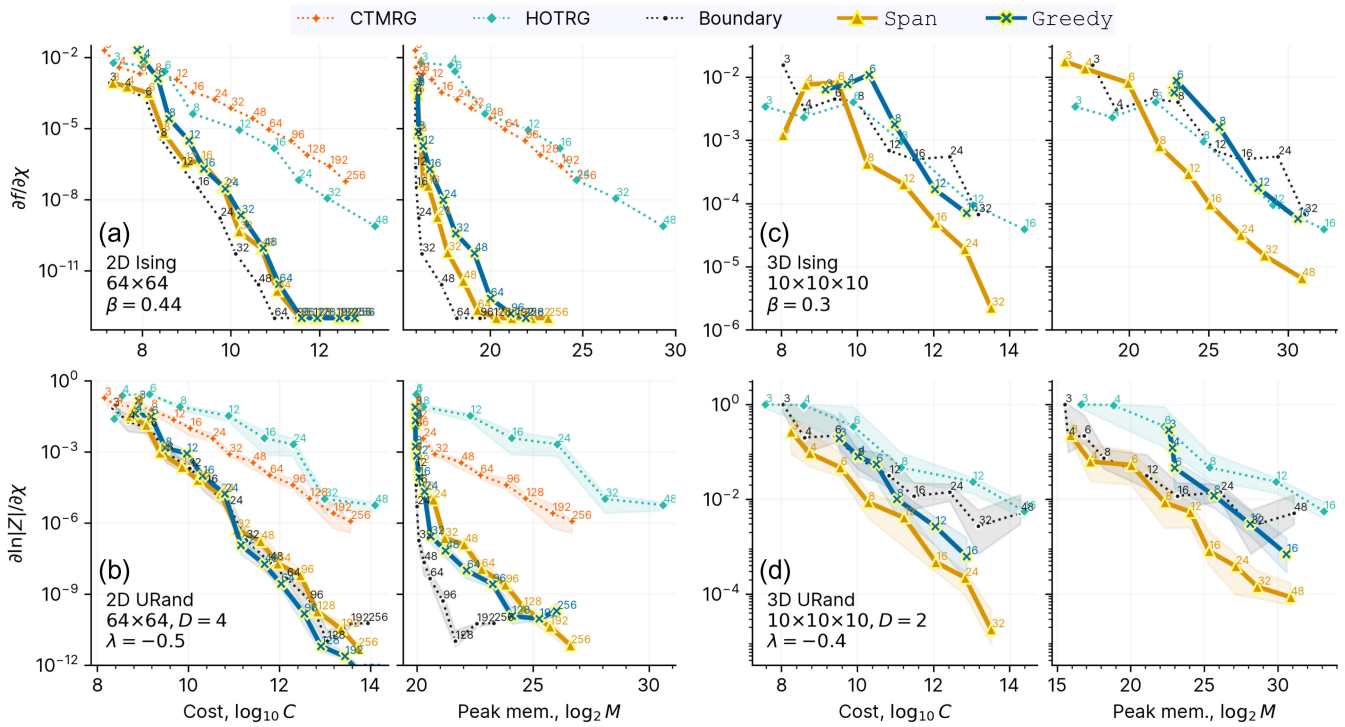

FIG. 13. Error versus cost of hyperoptimized approximate contraction using optimized `Span`, `Greedy`, and `Agglom` trees, in comparison to boundary contraction, CTMRG, and HOTRG, for large TNs where no exact reference is available. As a proxy for error, we monitor the rate of change of the log contraction value or free energy with bond dimension, $d\ln|Z|/d\chi$, $df/d\chi$, respectively. We plot this against both contraction cost and peak memory, where the trees have been optimized separately for each. The hyperoptimized methods use gauging settings of $r = 6$ in 2D and $r = 2$ in 3D, both with early compression. Four different lattice and model combinations are shown: (a) 2D Ising model, (b) 2D URrand model, (c) 3D Ising model, and (d) 3D URrand model.

computed by exact TN contraction. In Fig. 13 we consider the same models, but now for problem sizes too large for exact contraction. In these cases, we use $d\ln|Z|/d\chi$ and $df/d\chi$ as a metric of the convergence of the calculation.

From these plots we observe a few features. In the 2D models, the optimized `Span`, `Greedy`, and standard boundary contraction algorithms generally all achieve quite similar performance, and are the best performing algorithms. (We note that although `Span` shows a consistent advantage over boundary contraction in the error versus χ plots in Sec. III C, it does not do so when the overall cost or memory is considered, because this depends on additional details besides χ, such as the number of large tensors, order in which they contracted, etc.) CTMRG, HOTRG, and `Agglom` also perform similarly, and all perform much worse than the `Span`, `Greedy`, and standard boundary contraction algorithms on this regular 2D lattice.

In 3D, the PEPS boundary, CTMRG, and HOTRG all perform quite poorly, while `Span` performs well. `Greedy` performs well in the smaller examples, but degrades in the larger lattice, presumably again because of the limited contraction tree space generated by the `Greedy` algorithm. As noted in Sec. III C, hyperoptimized `Span` trees choose a quite different contraction path than the PEPS boundary algorithm, while still taking advantage of the boundary, and this is key to the improved performance.

Taken in total, the comparisons in the last three subsections illustrate how the optimized approximate contraction trees are competitive with, and can even exceed, the performance of standard contraction strategies in the simple lattices studied in many-body physics applications.

E. Comparison to another strategy for general graphs

We next turn to a comparison of our hyperoptimized approximate contraction strategy to another recently proposed technique for arbitrary graphs. Reference [52] proposed an algorithm to automatically contract arbitrary geometry tensor networks with a good performance across a range of graphs. For convenience we refer to that algorithm as `CATN`. As we cannot trace through `CATN` in the same way as our previous performance comparisons, we measure the contraction time directly on a single CPU. Although `CATN` is formulated in a geometry independent manner, a critical difference with the current work is that `CATN` does not optimize over families of approximate contraction trees.

In Fig. 14 we compare `CATN` against hyperoptimized `Span` trees for the 2D or 3D Ising model at approximately the critical point, as a function of accuracy versus contraction time. For the `Span` trees, we sweep over χ for

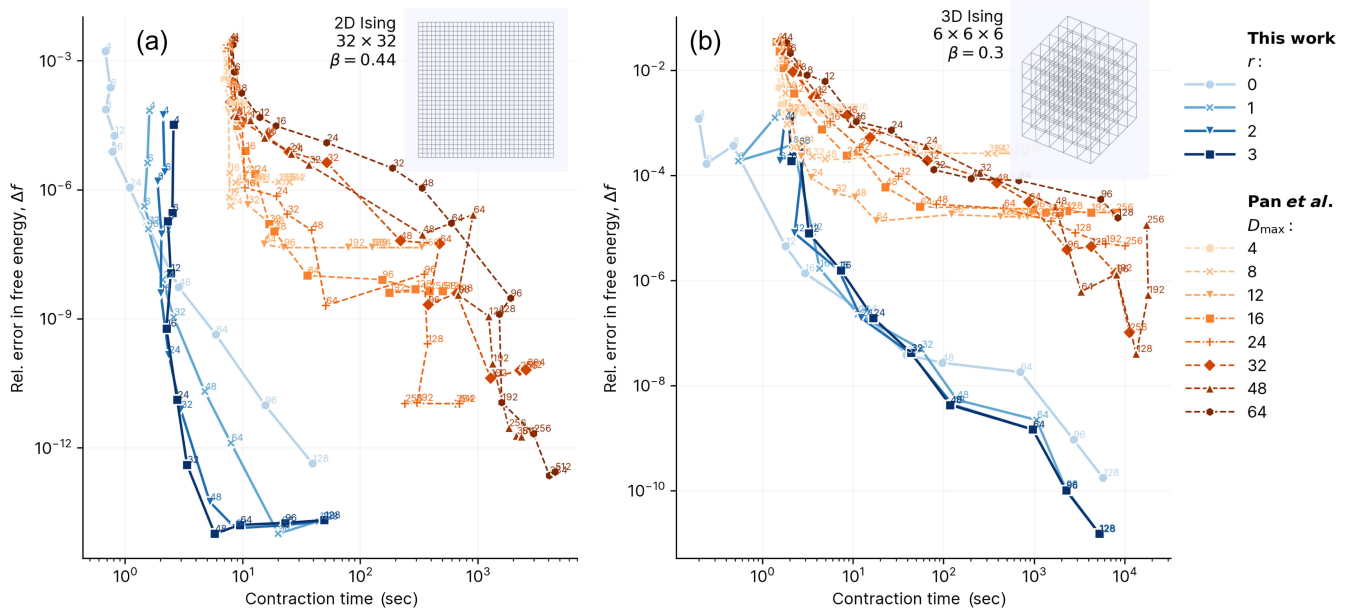

FIG. 14. Performance comparison of hyperoptimized approximate contraction (current work) and the algorithm of Pan *et al.* [52] for computing the free energy of the Ising model at approximately the critical point of (a) a square lattice and (b) a cubic lattice. For both algorithms χ is varied and the points are labeled with the value. The insets show the geometry and specific sizes of the lattices.

different choices of tree gauge distance r (i.e., each line corresponds to one r, while χ is swept over). The performance of `CATN` is considered as a function of the two bond dimensions $D_{\max}$ and χ (each line corresponds to a given $D_{\max}$, while χ is swept over). We do not include the time to find the tree for the hyperoptimized contraction since one generally reuses this many times. However, as a rough guide, for the lattices in Fig. 14 the search converges to a good tree in 10–20 sec. Further details and comparisons are in the SM [57]. We see clearly that in both the 2D and 3D cases [Figs. 14(a) and 14(b), respectively] the hyperoptimized `Span` trees achieve a better accuracy versus contraction time trade-off than the `CATN` algorithm. Given that `CATN` itself has an ordering of compressions, an interesting question is to what extent the strategy of `CATN` might also be optimized.

IV. POWER OF HYPEROPTIMIZED APPROXIMATE CONTRACTION

We now illustrate the power of the hyperoptimized approximate contraction protocol defined above in a further range of interesting problems.

A. Ising partition function on the pyrochlore lattice

We consider a tensor network contraction corresponding to the Ising partition function on large, finite, pyrochlore lattices with up to 4000 sites. (We consider the version of the model where all spins are aligned along the *same* axis; see Ref. [69].) The highly frustrated geometry makes it harder to compute a low complexity contraction path for this tensor network than in simpler lattices. In Fig. 15(a) we show the peak memory for the optimized `Span` algorithm as a function of side length L. The total lattice size is $L \times L \times L \times 4$; thus, the largest calculation ($L = 10$) is a contraction of 4000 tensors. For $L > 6$ we see the peak memory starts to saturate. By fitting $f(\chi) = A + B/\chi$ to our data for parameters A and B we can accurately estimate both the free energy and its error as the fitted value and square root variance of the parameter $A = f(\infty)$. We show the result of this in Fig. 15(b), in the vicinity of the critical point [70,71]. We note that while Ising systems like this can be studied using Monte Carlo techniques, the partition function itself is tricky to estimate, requiring methods [72] beyond the standard Metropolis algorithm [73]. Figure 15(d) shows the second derivative of $(1/N)\ln Z$ with respect to inverse temperature, $\partial^2(-\beta f)/\partial\beta^2$, which displays a growing peak as a function of system length L, illustrating the critical point.

The largest *exactly* contractable tensor network corresponds to size $L = 6$; it is visualized in the inset Fig. 15(c). For this size we can investigate the free energy error of the approximate contraction scheme Δf, and this is shown in Fig. 15(e). We see that increasing χ reliably decreases the error, and for the largest χ considered, the relative free energy is only 10^{-4}.

B. Random 3-regular graphs and dimer coverings

We next study a problem defined on random 3-regular graphs. Here the `Agglom` algorithm performs best, and we show the resulting complexities in terms of peak memory M in Fig. 16(a). Here, because the length of loops in the

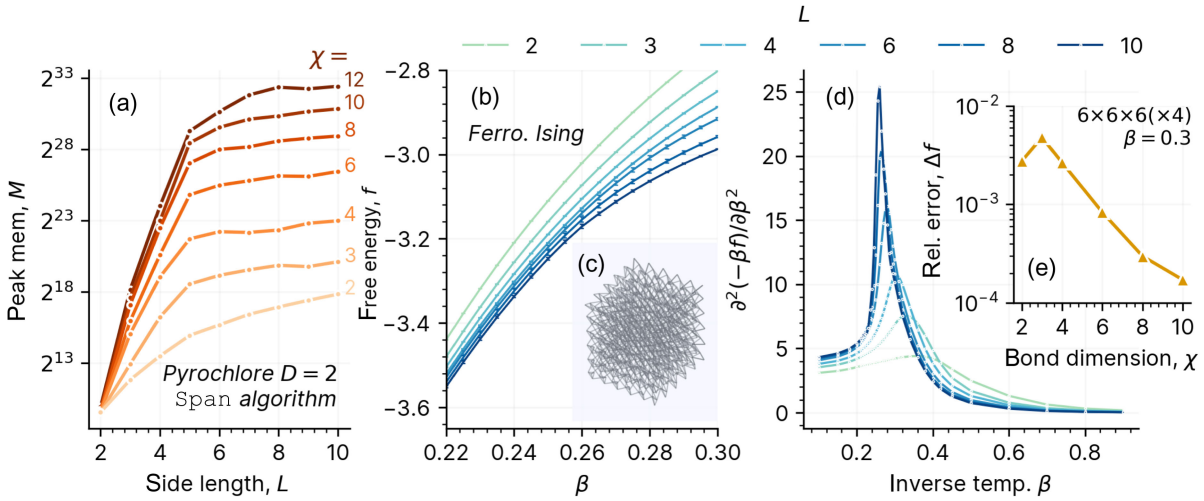

FIG. 15. Approximate contraction of the ferromagnetic Ising model on the pyrochlore lattice. (a) Peak memory M of the Span algorithm on the pyrochlore lattice with $D = 2$ as function of side length L and χ. (b) The free energy f near the critical point, estimated by extrapolating approximate contractions in χ. A tree-gauge distance of $r = 2$ was used and the error bars show fit uncertainty. (c) Example instance of the pyrochlore geometry for $L = 6$ corresponding to 864 sites. (d) The second derivative of $-\beta f$ with respect to β, showing a diverging peak around the critical point for increasing L. (e) Relative error in free energy near the critical point as a function of χ for $L = 6$ and $r = 2$.

graph grows with increasing number of vertices $|V|$, we still find an exponential scaling, even when compressing to fixed χ. Nonetheless, we can usefully push much beyond exactly contractible limit of $|V| \sim 300$ [illustrated in Fig. 16(c)]. To study the accuracy we consider the problem of counting dimer coverings on these graphs. This is equivalent to so-called positive Sharp-1-IN-3SAT [36,74,75]. Each edge (i.e., index) is considered a potential dimer, and by placing the tensor

$$T_{i,j,k} = \begin{cases} 1 & \text{if } i+j+k = 1 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

on each vertex, we enforce that every tensor be "covered" by a single dimer only for any configuration to be valid. The decision version of this problem is NP-complete [76,77], and on random 3-regular graphs specifically the problem is known to be close, but just on the satisfiable side, in terms of ratio of clauses to variables, of the hardest regime [74,75]. The contraction of the above tensor network gives the number of configurations Z at zero temperature and a corresponding "residual" entropy, $S = \ln Z$. We plot S in Fig. 16(b). Considering the entropy per site $s/|V|$ by performing a least squares fit with a quadratic function of inverse size $1/|V|$ and bond dimension $1/\chi$,

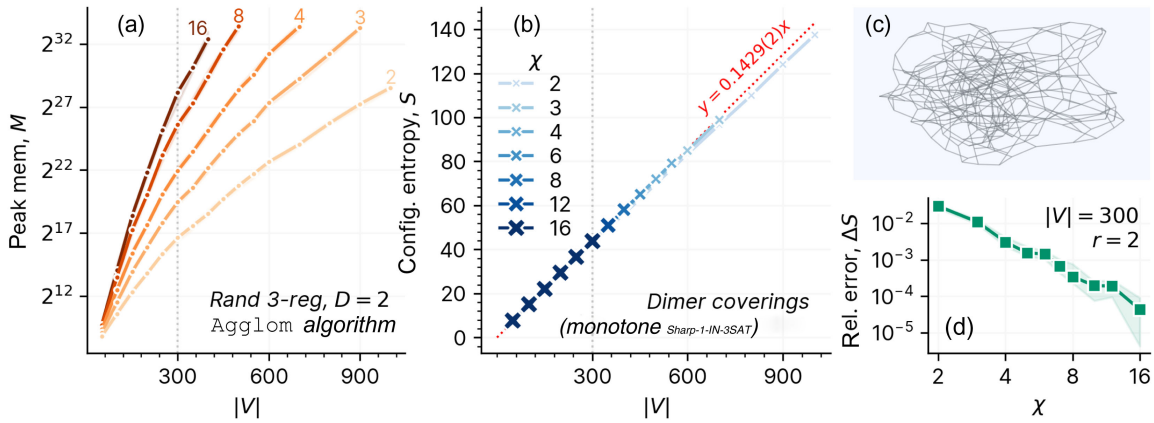

FIG. 16. Approximate contraction of dimer covering counting on random 3-regular graphs. Quantities are averaged over 20 instances. (a) Peak memory M of the Agglom algorithm on random 3-regular graph instances with $D = 2$ as a function of number of vertices $|V|$. (b) Configuration entropy, $S = \ln W$, where W is the number of valid configurations as a function of $|V|$ and χ. The red dotted line shows the constant from a least squares fit to a quadratic function of inverse $|V|$ and χ (see main text). (c) Example random regular graph for $|V| = 300$. (d) Relative error in S as a function of χ for $|V| = 300$.

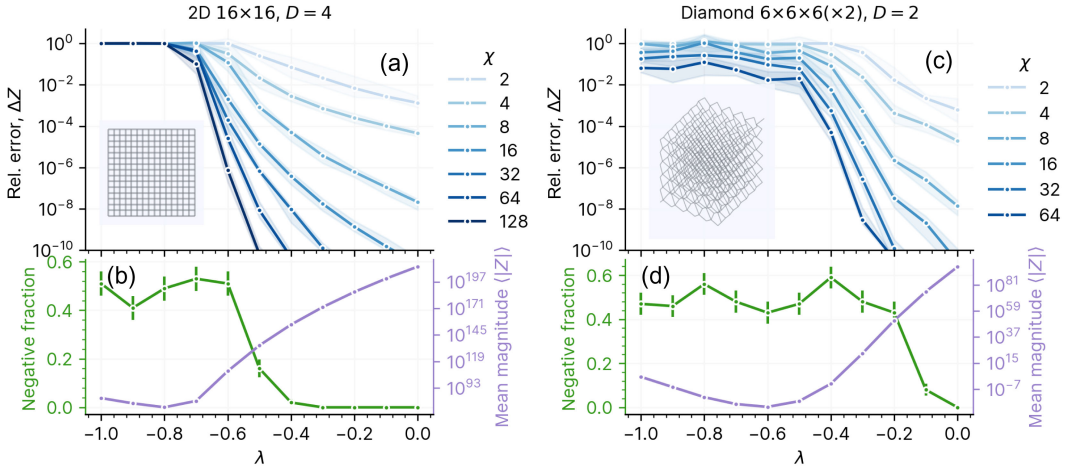

FIG. 17. Hardness transition in approximately contracting tensor networks with random uniform entries $\in [\lambda, 1]$. (a) Relative error ΔZ in approximately contracted value of the URand model on the square lattice using the `Greedy` algorithm as a function of λ and χ with $r = 2$. Line and bands show median and interquartile range across 20 instances. (b) Distribution of actual values Z for the square URand model in terms of fraction of negative instances (green, left-hand axis) and average absolute magnitude (purple, right-hand axis). Error bars denote error on mean. (c) Relative error ΔZ in approximately contracted value of the URand model on the diamond lattice using the `Greedy` algorithm as a function of λ and χ with $r = 2$. Line and bands show median and interquartile range across 20 instances. (d) Distribution of actual values Z for the diamond URand model in terms of fraction of negative instances (green, left-hand axis) and average absolute magnitude (purple, right-hand axis). Error bars denote error on mean.

$$s(|V|,\chi) = s_\infty + \frac{c_1}{|V|^2} + \frac{c_2}{|V|} + \frac{c_3}{|V|\chi} + \frac{c_4}{\chi} + \frac{c_5}{\chi^2}, \quad (6)$$

with fitted parameters, $s_\infty, c_{1\ldots5}$, we can estimate the infinite size entropy per site as $s_\infty = 0.1429(2)$. The theoretical value of this can be computed using Ref. [78] when $|V| \gtrsim 5 \times 10^{11}$ (see the SM [57]), yielding 0.1438, suggesting a small systematic error remains from the finite size. In Fig. 16(d) we consider the relative error in S when compared to exact contraction results for $|V| = 300$, where again we see that increasing χ reliably improves the error.

C. Hardness transition in random tensor networks

The final problem we consider is one where the hardness derives from the tensor entries themselves rather than the geometry. We take the URand model—with tensor entries sampled uniformly $\in [\lambda, 1]$—and consider two lattices which are amenable to contraction with relatively large χ, the square and diamond lattices. We take sizes 16×16 with $D = 4$ and $6 \times 6 \times 6(\times 2)$ with $D = 2$, respectively, both of which are at the limit of what is contractible exactly. In Fig. 17(a) we show the relative error in the approximately contracted value ΔZ as a function of λ across 20 random instances. There is a clear transition in hardness at $\lambda \sim -0.7$—above this even moderate χ is sufficient to contract the tensor network with very good accuracy. Below this, however, there is no improvement to the error at all with increasing χ; the contracted value remains essentially impossible to approximate. An obvious question is how does Z itself change with λ? In Fig. 17(b) we show the fraction of instances whose exact value Z is negative, as well as the average magnitude of Z. The problem varies from smaller magnitude values (compared to the total number of terms in the sum, $\sim 10^{300}$) evenly split between negative and positive, to large always positive values. We also consider the same model but embedded in a 3D diamond geometry in Fig. 17(c). The same transition in hardness occurs at a slightly different value of λ. In the hard regime, there remains some small ability to approximate Z with large χ, probably as this is approaching exact contraction. The complexity of contraction of the random tensor network is thus closely related to the positive nature of the tensor entries. This is likely related to the conjectured low entanglement of typical positive tensor networks [79], as well as the hardness of approximating complex valued Ising models [80–82].

V. CONCLUSIONS

We have introduced a framework for approximate contractions of tensor networks defined on arbitrary graphs, based on hyperoptimizing over ordered contraction trees with compression steps. In particular, our work attempts to optimize over the many choices and components in such an algorithm, ranging from the manner in which compressions are performed to the sequence and ordering of compression and contractions. Interestingly, we observe that by minimizing a cost function associated with memory or computational cost, we simultaneously generate an approximate contraction tree that yields small contraction error. In many cases, we find that the optimization produces significantly cheaper and more accurate contraction strategies

than handcrafted approximate contraction algorithms, even in well-studied regular lattices. While we cannot claim that our final algorithm is optimal, the purpose of the framework is to allow an optimization over compression strategies, and such an optimization can be extended should, for example, other metrics of approximate contraction quality be introduced. We envisage that the many constituent parts of our protocol can be separately improved in future works.

We have discussed different regimes of computational advantage for approximate contraction over exact contraction. Firstly, for locally connected graphs and "nonhard" tensor entries, we expect approximate contraction to display an exponential benefit over exact contraction, as shown here for the pyrochlore lattice. Secondly, for certain geometries with long-range interactions, we expect approximate contraction to still scale exponentially, but with a usefully reduced prefactor, as shown here for 3-regular random graphs. Finally, we expect some classes of tensor entries to be essentially impossible to approximately contract, regardless of geometry, as shown here for certain distributions of random tensors. Whether the latter result corresponds to the hardness of contraction expected for generic random quantum circuits is an interesting question. Similarly, the application of these techniques to quantum circuit and ansatz expectation values is a natural direction that we leave for future work.

ACKNOWLEDGMENTS

We thank Stefanos Kourtis and Pan Zhang for helpful comments on this manuscript, and Hitesh Changlani for help with understanding the pyrochlore lattice. The development of the contraction tree optimization algorithms was supported by the U.S. National Science Foundation through Grant No. 1931328. The approximate contraction algorithms were developed with support from the U.S. National Science Foundation through Grant No. 2102505. J. G. acknowledges support through a gift from Amazon Web Services Inc. G. K.-L. C. is partially supported as a Simons Investigator in Physics.

[1] I. Markov and Y. Shi, *Simulating quantum computation by contracting tensor networks*, SIAM J. Comput. **38**, 963 (2008).

[2] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff, *Breaking the 49-qubit barrier in the simulation of quantum circuits*, arXiv:1710.05867.

[3] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, *Simulation of low-depth quantum circuits as complex undirected graphical models*, arXiv:1712.05384.

[4] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, *Classical simulation of intermediate-size quantum circuits*, arXiv:1805.01450.

[5] B. Villalonga, S. Boixo, B. Nelson, C. Henze, E. Rieffel, R. Biswas, and S. Mandrà, *A flexible high-performance simulator for the verification and benchmarking of quantum circuits implemented on real hardware*, npj Quantum Inf. **5**, 86 (2019).

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

[7] C. Huang, F. Zhang, M. Newman, J. Cai, X. Gao, Z. Tian, J. Wu, H. Xu, H. Yu, B. Yuan, M. Szegedy, Y. Shi, and J. Chen, *Classical simulation of quantum supremacy circuits*, arXiv:2005.06787.

[8] G. Kalachev, P. Panteleev, and M.-H. Yung, *Recursive multi-tensor contraction for XEB verification of quantum circuits*, arXiv:2108.05665.

[9] F. Pan and P. Zhang, *Simulation of quantum circuits using the big-batch tensor network method*, Phys. Rev. Lett. **128**, 030501 (2022).

[10] S. R. White, *Density-matrix algorithms for quantum renormalization groups*, Phys. Rev. B **48**, 10345 (1993).

[11] S. R. White, *Density matrix formulation for quantum renormalization groups*, Phys. Rev. Lett. **69**, 2863 (1992).

[12] V. Murg, F. Verstraete, and J. I. Cirac, *Variational study of hard-core bosons in a two-dimensional optical lattice using projected entangled pair states*, Phys. Rev. A **75**, 033605 (2007).

[13] F. Verstraete and J. I. Cirac, *Renormalization algorithms for quantum-many body systems in two and higher dimensions*, arXiv:cond-mat/0407066.

[14] G. Vidal, *Classical simulation of infinite-size quantum lattice systems in one spatial dimension*, Phys. Rev. Lett. **98**, 070201 (2007).

[15] H. C. Jiang, Z. Y. Weng, and T. Xiang, *Accurate determination of tensor network state of quantum lattice models in two dimensions*, Phys. Rev. Lett. **101**, 090603 (2008).

[16] E. Stoudenmire and S. R. White, *Studying two-dimensional systems with the density matrix renormalization group*, Annu. Rev. Condens. Matter Phys. **3**, 111 (2012).

[17] P. C. G. Vlaar and P. Corboz, *Simulation of three-dimensional quantum systems with projected entangled-pair states*, Phys. Rev. B **103**, 205137 (2021).

[18] T. Nishino and K. Okunishi, *Corner transfer matrix renormalization group method*, J. Phys. Soc. Jpn. **65**, 891 (1996).

[19] T. Nishino and K. Okunishi, *Corner transfer matrix algorithm for classical renormalization group*, J. Phys. Soc. Jpn. **66**, 3040 (1997).

[20] M. Levin and C. P. Nave, *Tensor renormalization group approach to two-dimensional classical lattice models*, Phys. Rev. Lett. **99**, 120601 (2007).

[21] R. Orus and G. Vidal, *Simulation of two-dimensional quantum systems on an infinite lattice revisited: Corner transfer matrix for tensor contraction*, Phys. Rev. B **80**, 094403 (2009).

[22] Z. Y. Xie, H. C. Jiang, Q. N. Chen, Z. Y. Weng, and T. Xiang, *Second renormalization of tensor-network states*, Phys. Rev. Lett. **103**, 160601 (2009).

[23] L. Vanderstraeten, B. Vanhecke, and F. Verstraete, *Residual entropies for three-dimensional frustrated spin systems with tensor networks*, Phys. Rev. E **98**, 042145 (2018).

[24] H.-H. Zhao, Z.-Y. Xie, T. Xiang, and M. Imada, *Tensor network algorithm by coarse-graining tensor renormalization on finite periodic lattices*, Phys. Rev. B **93**, 125115 (2016).

[25] J.-G. Liu, L. Wang, and P. Zhang, *Tropical tensor network for ground states of spin glasses*, Phys. Rev. Lett. **126**, 090506 (2021).
[26] A. J. Ferris and D. Poulin, *Tensor networks and quantum error correction*, Phys. Rev. Lett. **113**, 030501 (2014).
[27] S. Bravyi, M. Suchara, and A. Vargo, *Efficient algorithms for maximum likelihood decoding in the surface code*, Phys. Rev. A **90**, 032326 (2014).
[28] D. K. Tuckett, A. S. Darmawan, C. T. Chubb, S. Bravyi, S. D. Bartlett, and S. T. Flammia, *Tailoring surface codes for highly biased noise*, Phys. Rev. X **9**, 041031 (2019).
[29] C. T. Chubb and S. T. Flammia, *Statistical mechanical models for quantum codes with correlated noise*, Ann. Inst. Henri Poincaré **8**, 269 (2021).
[30] C. T. Chubb, *General tensor network decoding of 2D Pauli codes*, arXiv:2101.04125.
[31] J. P. Bonilla Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, *The XZZX surface code*, Nat. Commun. **12**, 2172 (2021).
[32] T. Farrelly, R. J. Harris, N. A. McMahon, and T. M. Stace, *Tensor-network codes*, Phys. Rev. Lett. **127**, 040507 (2021).
[33] N. Schuch, M. M. Wolf, F. Verstraete, and J. I. Cirac, *Computational complexity of projected entangled pair states*, Phys. Rev. Lett. **98**, 140506 (2007).
[34] A. García-Sáez and J. I. Latorre, *An exact tensor network for the 3SAT problem*, Quantum Inf. Comput. **12**, 283 (2012).
[35] J. D. Biamonte, J. Morton, and J. Turner, *Tensor network contractions for #SAT*, J. Stat. Phys. **160**, 1389 (2015).
[36] S. Kourtis, C. Chamon, E. Mucciolo, and A. Ruckenstein, *Fast counting with tensor networks*, SciPost Phys. **7**, 060 (2019).
[37] J. M. Dudek, L. Duenas-Osorio, and M. Y. Vardi, *Efficient contraction of large tensor networks for weighted model counting through graph decompositions*, arXiv:1908.04381.
[38] J. M. Dudek and M. Y. Vardi, *Parallel weighted model counting with tensor networks*, arXiv:2006.15512.
[39] N. de Beaudrap, A. Kissinger, and K. Meichanetzidis, *Tensor network rewriting strategies for satisfiability and counting*, Electron. Proc. Theor. Comput. Sci. **340**, 46 (2021).
[40] A. J. Gallego and R. Orus, *Language design as information renormalization*, SN Comput. Sci. **3**, 140 (2022).
[41] V. Pestun and Y. Vlassopoulos, *Tensor network language model*, arXiv:1710.10248.
[42] T.-D. Bradley, E. M. Stoudenmire, and J. Terilla, *Modeling sequences with quantum states: A look under the hood*, Mach. Learn. Sci. Technol. **1**, 035008 (2020).
[43] K. Meichanetzidis, A. Toumi, G. de Felice, and B. Coecke, *Grammar-aware sentence classification on quantum computers*, Quantum Mach. Intell. **5**, 10 (2023).
[44] F. Verstraete, V. Murg, and J. I. Cirac, *Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems*, Adv. Phys. **57**, 143 (2008).
[45] R. Orús, *Tensor networks for complex quantum systems*, Nat. Rev. Phys. **1**, 538 (2019).
[46] B. Dittrich, F. C. Eckert, and M. Martin-Benito, *Coarse graining methods for spin net and spin foam models*, New J. Phys. **14**, 035008 (2012).
[47] G. Evenbly and G. Vidal, *Tensor network renormalization*, Phys. Rev. Lett. **115**, 180405 (2015).
[48] S. Yang, Z.-C. Gu, and X.-G. Wen, *Loop optimization for tensor network renormalization*, Phys. Rev. Lett. **118**, 110504 (2017).
[49] M. Bal, M. Mariën, J. Haegeman, and F. Verstraete, *Renormalization group flows of Hamiltonians using tensor etworks*, Phys. Rev. Lett. **118**, 250602 (2017).
[50] S.-J. Ran, E. Tirrito, C. Peng, X. Chen, L. Tagliacozzo, G. Su, and M. Lewenstein, *Tensor Network Contractions: Methods and Applications to Quantum Many-Body Systems* (Springer Nature, Cham, Switzerland, 2020), 10.1007/978-3-030-34489-4æ.
[51] A. Jermyn, *Automatic contraction of unstructured tensor networks*, SciPost Phys. **8**, 005 (2020).
[52] F. Pan, P. Zhou, S. Li, and P. Zhang, *Contracting arbitrary tensor networks: General approximate algorithm and applications in graphical models and quantum circuit simulations*, Phys. Rev. Lett. **125**, 060503 (2020).
[53] There is a minor effect on memory.
[54] G. Evenbly and G. Vidal, *Tensor network renormalization*, Phys. Rev. Lett. **115**, 180405 (2015).
[55] M. Hauru, C. Delcamp, and S. Mizera, *Renormalization of tensor networks using graph-independent local truncations*, Phys. Rev. B **97**, 045111 (2018).
[56] S. Yang, Z.-C. Gu, and X.-G. Wen, *Loop optimization for tensor network renormalization*, Phys. Rev. Lett. **118**, 110504 (2017).
[57] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevX.14.011009 for detailed information about the implementation and application of the hyperoptimized approximate contraction method.
[58] G. Evenbly, *Gauge fixing, canonical forms, and optimal truncations in tensor networks with closed loops*, Phys. Rev. B **98**, 085155 (2018).
[59] M. Hauru, C. Delcamp, and S. Mizera, *Renormalization of tensor networks using graph independent local truncations*, Phys. Rev. B **97**, 045111 (2018).
[60] L. Wang and F. Verstraete, *Cluster update for tensor network states*, arXiv:1110.4362.
[61] P. Corboz, T. M. Rice, and M. Troyer, *Competing states in the t-J model: Uniform d-wave state versus stripe state*, Phys. Rev. Lett. **113**, 046402 (2014).
[62] S. Iino, S. Morita, and N. Kawashima, *Boundary tensor renormalization group*, Phys. Rev. B **100**, 035449 (2019).
[63] S. Schlag, *High-Quality Hypergraph Partitioning*, Ph.D. thesis, Karlsruhe Institute of Technology, 2020.
[64] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, *High-quality hypergraph partitioning*, ACM J. Exp. Algorithmics **27**, 1 (2023).
[65] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019* (Association for Computing Machinery (ACM), New York, 2019).
[66] J. Rapin and O. Teytaud, *nevergrad—A gradient-free optimization platform*, https://GitHub.com/FacebookResearch/Nevergrad.
[67] T. Nishino and K. Okunishi, *Corner transfer matrix renormalization group method*, J. Phys. Soc. Jpn. **65**, 891 (1996).

[68] Z.-Y. Xie, J. Chen, M.-P. Qin, J. W. Zhu, L.-P. Yang, and T. Xiang, *Coarse-graining renormalization by higher-order singular value decomposition*, Phys. Rev. B **86**, 045139 (2012).

[69] S. Bramwell and M. Harris, *Frustration in Ising-type spin models on the pyrochlore lattice*, J. Phys. Condens. Matter **10**, L215 (1998).

[70] A. L. Passos, D. F. de Albuquerque, and J. B. S. Filho, *Representation and simulation for pyrochlore lattice via Monte Carlo technique*, Physica (Amsterdam) **450A**, 541 (2016).

[71] K. Soldatov, K. Nefedev, Y. Komura, and Y. Okabe, *Large-scale calculation of ferromagnetic spin systems on the pyrochlore lattice*, Phys. Lett. A **381**, 707 (2017).

[72] F. Wang and D. P. Landau, *Efficient, multiple-range random walk algorithm to calculate the density of states*, Phys. Rev. Lett. **86**, 2050 (2001).

[73] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *Equation of state calculations by fast computing machines*, J. Chem. Phys. **21**, 1087 (1953).

[74] J. Raymond, A. Sportiello, and L. Zdeborová, *Phase diagram of the 1-in-3 satisfiability problem*, Phys. Rev. E **76**, 011101 (2007).

[75] L. Zdeborová and M. Mézard, *Constraint satisfaction problems with isolated solutions are hard*, J. Stat. Mech. (2008) P12004.

[76] T. J. Schaefer, *The complexity of satisfiability problems*, in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC'78* (Association for Computing Machinery, New York, 1978), pp. 216–226.

[77] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W.H. Freeman & Co., San Francisco, 1979).

[78] B. Bollobás and B. D. McKay, *The number of matchings in random regular graphs and bipartite graphs*, J. Comb. Theory Ser. B **41**, 80 (1986).

[79] T. Grover and M. P. A. Fisher, *Entanglement and the sign structure of quantum states*, Phys. Rev. A **92**, 042308 (2015).

[80] L. A. Goldberg and H. Guo, *The complexity of approximating complex-valued Ising and Tutte partition functions*, arXiv:1409.5627.

[81] A. Galanis, L. A. Goldberg, and A. Herrera-Poyatos, *The complexity of approximating the complex-valued Ising model on bounded degree graphs*, arXiv.2105.00287.

[82] P. Buys, A. Galanis, V. Patel, and G. Regts, *Lee-Yang zeros and the complexity of the ferromagnetic Ising model on bounded-degree graphs*, Forum Math. Sigma **10**, e7 (2022).