

Hyper-optimized 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, USA 91125

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 hyper-optimization over the compression and contraction strategy itself to minimize error and cost. We demonstrate that our protocol outperforms both hand-crafted 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.

I. INTRODUCTION

Tensor network contraction, a summation over a product of multi-dimensional 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 “hyper-optimization”. 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 hyper-optimized 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 multi-dimensional array). We use lower indices to index into the tensor, e.g.

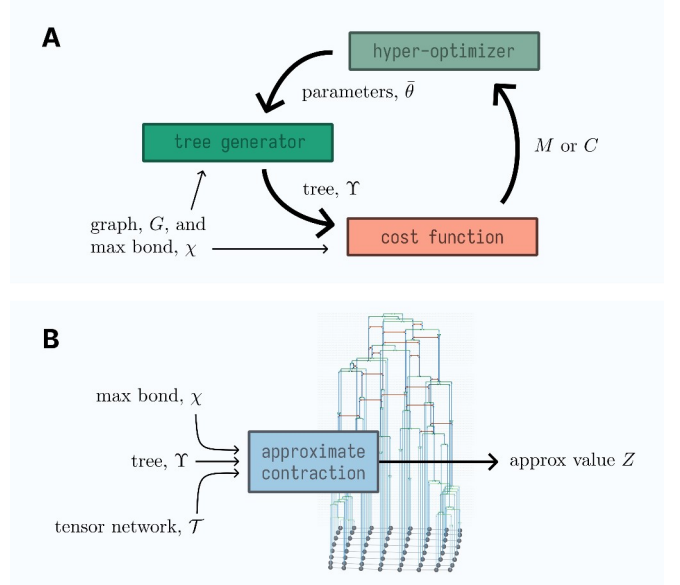


FIG. 1. Overview. The approximate contraction is specified by a sequence of contractions and compressions, expressed as an ordered tree. The strategy optimizes a cost-function over such trees. **A:** The hyper-optimization loop. Approximate contraction trees Y on the graph G are suggested by the tree generator. The tree characteristics are controlled by heuristic parameters θ and maximum bond dimension χ . The hyper-optimizer minimizes a cost-function M or C (peak memory or computational cost). **B:** The approximate contraction tree. The tensor network T is shown at the bottom. Moving upwards, 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 .

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

$$T_{\{e_{\text{out}}\}} = \sum_{\{e\}/\{e_{\text{out}}\}} \prod_v T_{\{e_v\}}^{[v]} \quad (1)$$

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),

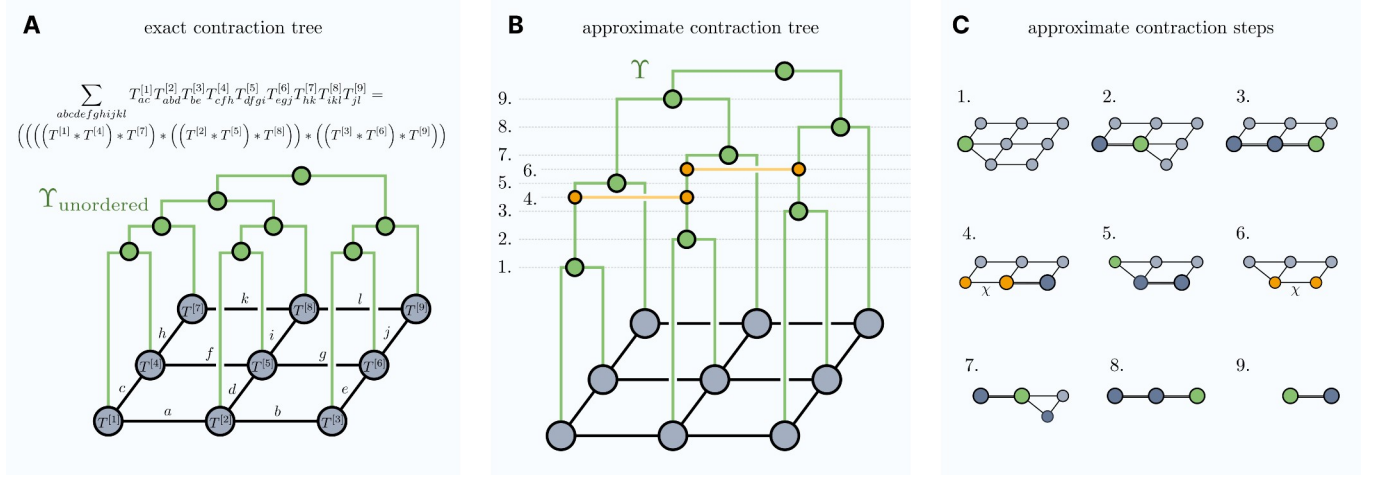


FIG. 2. **A.** Pairwise exact contraction of a tensor network, with the unordered contraction tree $\Upsilon_{\text{unordered}}$ indicating the contractions. Each intermediate (green node) corresponds to a pair of parentheses in the expression. **B.** An approximate contraction tree Υ for same network. 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.

with the bonds (edges) corresponding to the indices $\{e\}$. An examples of contraction is shown in Fig. 2A.

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 (hyper-optimization) 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 hyper-optimized 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 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 hyper-optimized 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. A 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. 2A, 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 does 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 $AB \approx \tilde{A}\tilde{B}$ where $\tilde{A}$ is of dimension $D \times \chi$ and $\tilde{B}$ 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. 3A), we can view the bonds as performing a matrix contraction (Fig. 3B), and use SVD to replace the connecting bonds by one of dimension χ (Fig. 3D). 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 section II C.

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 $\mathcal{Y}$ is shown in Fig. 2B, 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. 2C. 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 hyper-optimization strategy, where (families) of trees are parameterized by a small set of heuristic parameters, constituting a reduced dimensionality search space (described in Secs. II G, 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 hyper-optimized contraction trees. This is the subject of section 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 RG flow [56]. We discuss the corner double line (CDL) model and show that it is accurately contractible with our approximate strategy in the SI [57].

To aid in our discussion of the ingredients of the approxima-

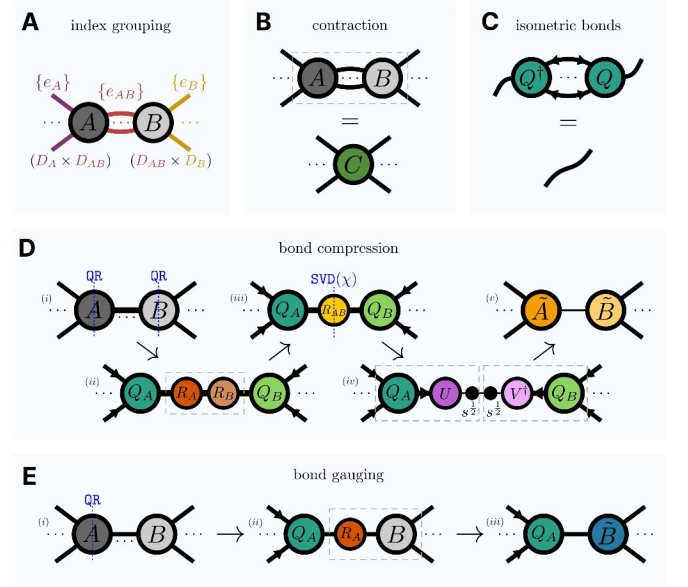


FIG. 3. Primitive tensor operators for approximate contraction. **A**: Grouping of indices into left, shared and right sets, giving a matricization of the product AB , with rank D_{AB} . **B** Graphical depiction of contracting two tensors $AB \rightarrow C$. **C** Graphical depiction of an isometric tensor, Q , such that when dimensions with incoming arrows are grouped $Q^T Q = 1$. **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 .

tion 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-random regular 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, (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

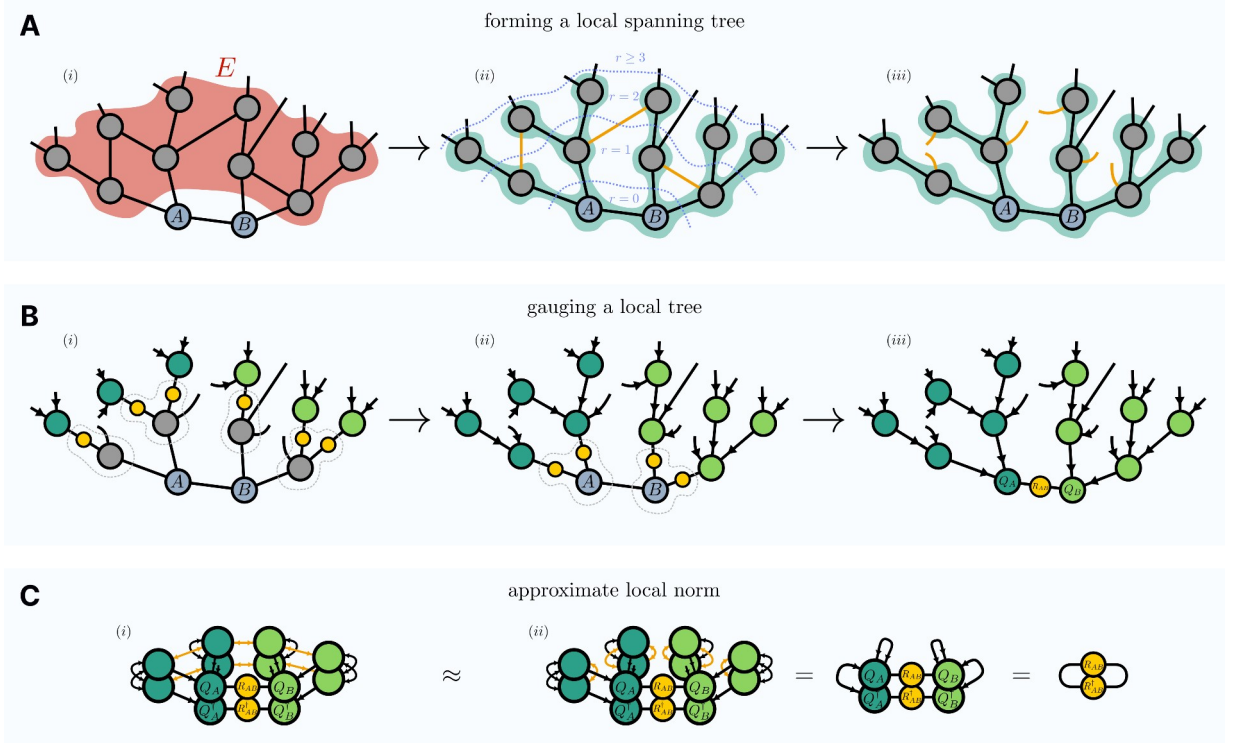


FIG. 4. Overview of the tree gauge for improving bond compression accuracy, suitable for arbitrary local geometry. **A**: Given the bond e_{AB} connecting tensors A and B , we want to take into account information from the surrounding environment E , shown in (i). In (ii) we form a local spanning tree (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 towards the bond e_{AB} , see Fig. 3E. 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 $(\text{Tr} R_{AB}^\dagger R_{AB})^{1/2}$. Performing a truncated SVD on R_{AB} is thus only r -locally optimal up to the presence of such loop bonds.

discussion of these models (as well as a treatment of corner double line models [56]) is in the SI [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. 3A). 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 decompositions of the matricized A, B , giving

$$AB = Q_A (R_A R_B) Q_B \quad (2)$$

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

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. 3C (detailed in the SI [57]). Then, we obtain the compressed $\bar{A}$,

$\bar{B}$ through the SVD of R_{AB} ,

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

truncating to X 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 T will contain additional tensors connected to A, B . We refer to the additional network of connected tensors as the environment E , with $T = \sum_{\{e\}/\{e_{out}\}} A_{\{e_A\}} B_{\{e_B\}} E_{\{e_E\}}$ (Fig. 4A). 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. 4A(ii)). Then, we can perform QR inwards from the leaves of the tree, pushing the R factors towards the bond (Fig. 4B (i)–(iii)). This is a type of gauging of the tensor network (i.e. it changes the tensors but does not change the contraction 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

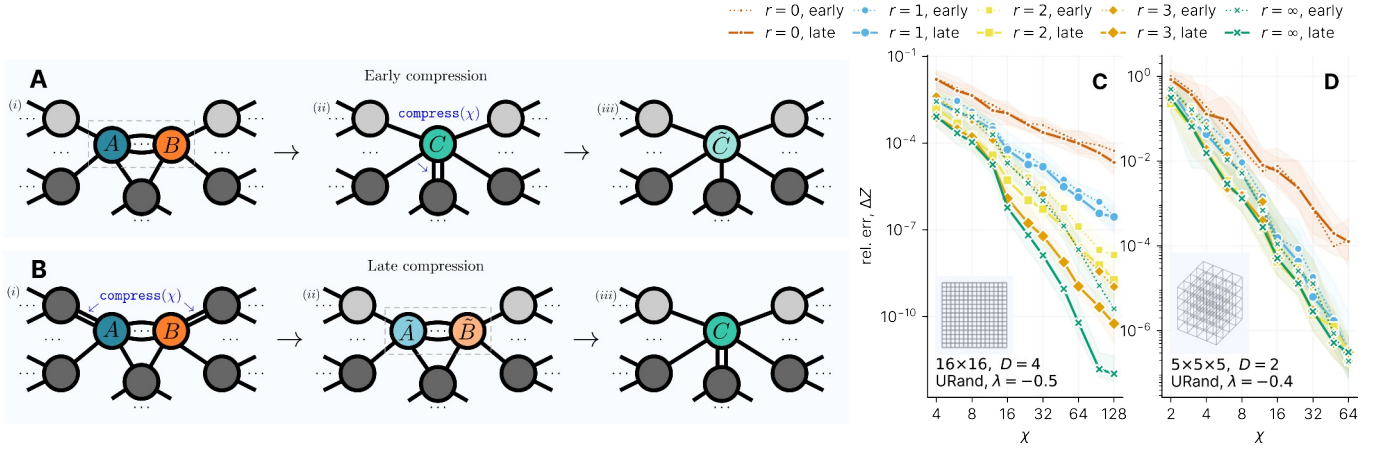


FIG. 5. **A:** Schematic of ‘early’ compression, where *after* each pairwise contraction, any shared bonds of total size $> \chi$ are truncated. **B:** Schematic of ‘late’ compression, where *before* each pairwise contraction, any shared bonds of total size $> \chi$ are truncated. **C:** Error ΔZ of 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 5×5 $D = 2$ tensor network with uniform random entries $\in [-0.4, 1]$. The 3D TN is contracted using a hyper-optimized Span tree.

e_{AB} . This results in a similar matricized $T = Q_A (R_{AB}) Q_B$ 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. 4B (iii)). Then, the truncated SVD of R_{AB} in (4) similarly achieves an optimal compression of e_{AB} with respect to error in T .

More generally, T may contain loops, which extend into the environment (Fig. 4A) 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 SI [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 neglecting loop correlations, see Fig. 4C. 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 $AB \approx AP_A P_B B$. 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. 5A. 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. 5B. By delaying the compression, more bonds/tensors in the environment are left uncompressed, which can potentially improve the quality of the contraction. However, late compression will also increase the cost/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. 5C, 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 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. 5D, we similarly compare early versus late compression using the tree gauge in a 3D lattice (using a hyper-optimized 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 so:

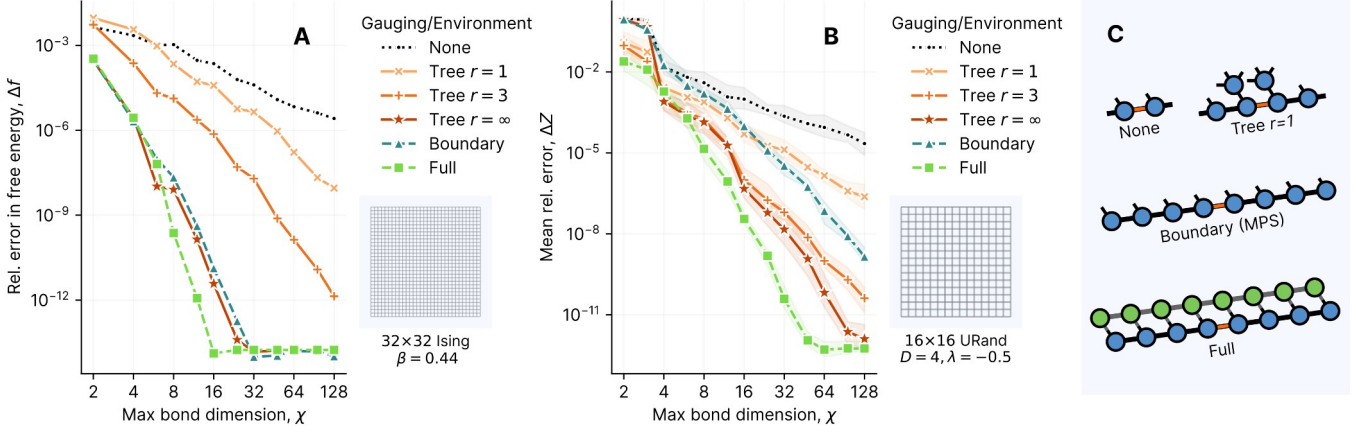


FIG. 6. **A**: Error in free energy, Δf , as a function of bond dimension, χ , for different gauging and environments for the 2D Ising model at the critical point. **B**: Contraction error, ΔZ , for the same settings but on a $D = 4$ URand model with $\lambda = -0.5$. All contractions contract from the 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 counter-propagating MPS of the same bond dimension, and the bond between AB is then truncated to minimize the error in $\text{Tr } BEA$. 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.

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/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 4 different gauges/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 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 SI for an illustration). 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 $Z = \text{Tr } BEA$ (where A, B, E have been matrixized). Using the eigenvector decomposition, $BEA = L \sigma R^T$, where L, R

are left, right eigenvectors respectively, then e_{AB} is optimally compressed by defining $\tilde{B} = L^T B, \tilde{A} = A R$, where L, 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.

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. 1B. The exact form of the

Algorithm 1 Approximate contraction

Input: tensor network T , ordered contraction tree Y , maximum bond dimension χ , tree gauge distance r , flag `compress_late` // i, j, k, l label tensors, $T^{[i]}, \dots$ in T .

```

for  $i, j \in Y$  do
  if compress_late then
    for  $l \in \text{NEIGHBORS}(T, i)$  do
      if  $\text{BONDSIZE}(T, i, l) > \chi$  and  $l \neq j$  then
        COMPRESS TREE GAUGE  $(\chi, r, T, i, l)$ 
      end if
    end for
  end if
  for  $l \in \text{NEIGHBORS}(T, j)$  do
    if  $\text{BONDSIZE}(T, j, l) > \chi$  and  $l \neq i$  then
      COMPRESS TREE GAUGE  $(\chi, r, T, j, l)$ 
    end if
  end for
end if
 $k \leftarrow \text{CONTRACT}(T, i, j)$ 
if not compress_late then
  for  $l \in \text{NEIGHBORS}(T, k)$  do
    if  $\text{BONDSIZE}(T, k, l) > \chi$  then
      COMPRESS TREE GAUGE  $(\chi, r, T, k, l)$ 
    end if
  end for
end if
end if
end for
Return:  $T$ 

```

inner functions is detailed in the SI [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 SI [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 parameterization 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 SI [57]). The input to the generators is only the tensor network graph, bond sizes and χ – the tensor entries are not considered.

The *Greedy* tree generator assigns a score to each bond in the T . It then chooses the highest scoring bond, generates a new T by simulating bond contraction and compression (i.e.

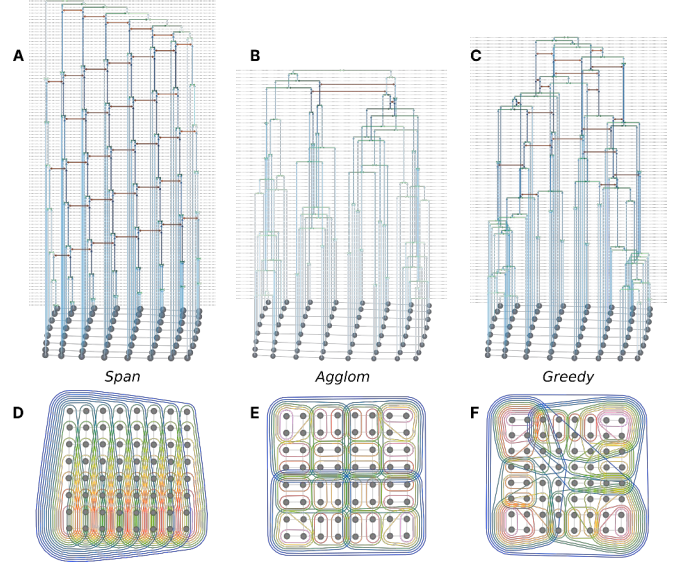


FIG. 7. Example approximate contraction trees for a 2D 8×8 square lattice TN. **A**, **B**, and **C** show the ordered contraction trees with compressions (orange, using the ‘late’ strategy) explicitly marked for the *Span*, *Agglom* and *Greedy* methods respectively. The computation proceeds from the bottom to top. **D**, **E**, and **F** show the same contraction trees as hierarchical communities, with the contraction ordering proceeding from the smallest pink bands to largest blue bands.

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 inwards. Contracting simultaneously along all the branches of the tree defines an effective contraction boundary, that sweeps towards the root. The algorithm begins by choosing either the most or least central node in the TN, i_0 , as an initial span, $S = \{i_0\}$. It then greedily expands to a connected node j , adding the contraction $(i, j) \rightarrow i$ in reverse order to the path, where $i \in S, j \notin S$. The algorithm repeats by then considering all neighbors of the newly expanded span $S \rightarrow S \cup \{j\}$. A few local quantities – connectivity to 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 T to find N/K roughly equal subgraphs. These subgraphs

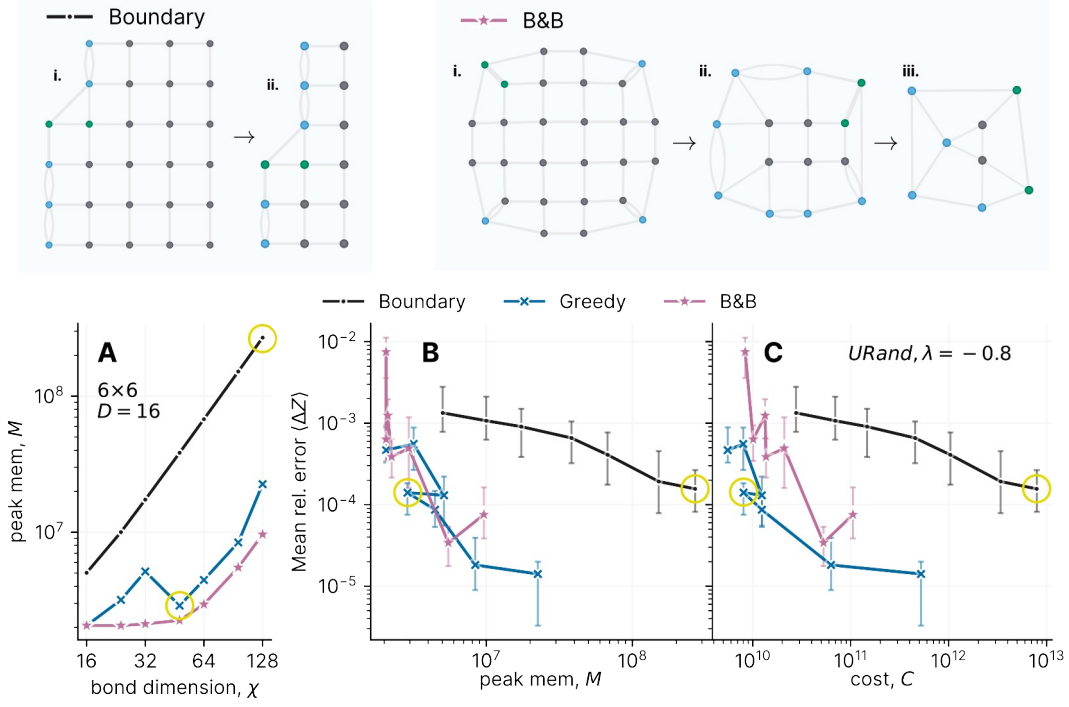


FIG. 8. Performance of hyper-optimized approximate contraction for a 6×6 square TN with all bonds of size $D = 16$. The TN is filled with uniformly random entries $\in [\lambda = -0.8, 1.0]$ (2D URand model). We test 3 different contraction trees: standard MPS boundary contraction, optimization over Greedy trees, brute force optimization over all approximate contraction trees (B&B). The upper insets show snapshots illustrating the ‘Boundary’ contraction, an example of Greedy, and the optimal ‘B&B’ contraction for $\chi = 32$. **A**: The peak memory usage, M , of the standard MPS ‘boundary’ method, compared to the Greedy and ‘B&B’ algorithms which have been optimized for each χ . **B** and **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 $\sigma = 1$. 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 $120\times$.

then define intermediate tensors, and the tensors within the sub-graph 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 boundary-like 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 4 corners inwards, 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.

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. 1A, with more detailed pseudo-code in the SI [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 3 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 hyper-optimization time is considered in the SI.

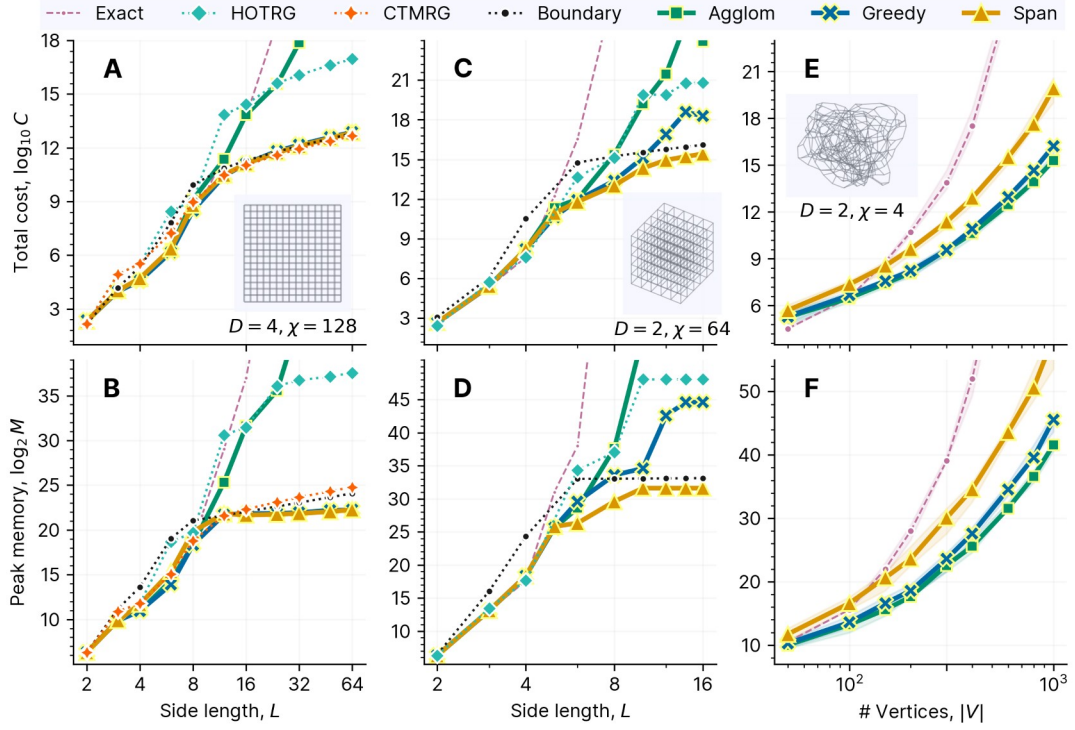


FIG. 9. Contraction peak memory and cost of approximate contraction trees in different families, versus exact contraction cost and CTMRG, HOTRG and boundary contraction algorithms, for different geometries (insets show sample geometry). The tree gauging distance is set to $r = 0$ for this comparison, with ‘early’ compression, and we also turn off gauging in the Boundary algorithm. The hyper-optimized trees are optimized for M and C separately. **A** and **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** and **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** and **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.

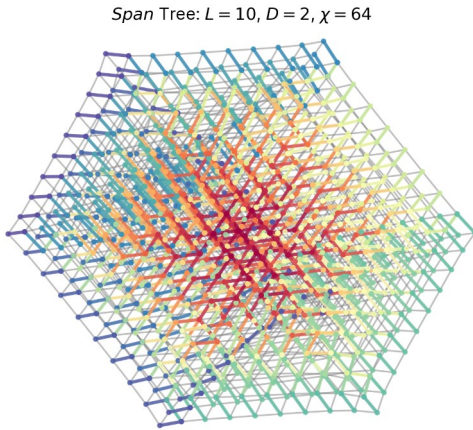


FIG. 10. Depiction of the contraction tree found by the Span algorithm, optimized for minimum floating point cost C , for a large 3D lattice. The colors of the highlighted edges and nodes indicate the stage of contraction they are involved in, running from blue (earliest) to red (latest).

I. Quality of hyper-optimization

To test the quality of the hyper-optimization and the tree search space, we first consider a small (but nonetheless non-trivial-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 (‘B&B’ – see SI); here we minimize peak memory M . In Fig. 8 we compare the performance of the hyper-optimized 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. 8A, hyper-optimizing over the space of Greedy trees produces performance quite similar to the B&B search and very different from the standard boundary contraction. This indicates that the hyper-optimization 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 B&B 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 er-

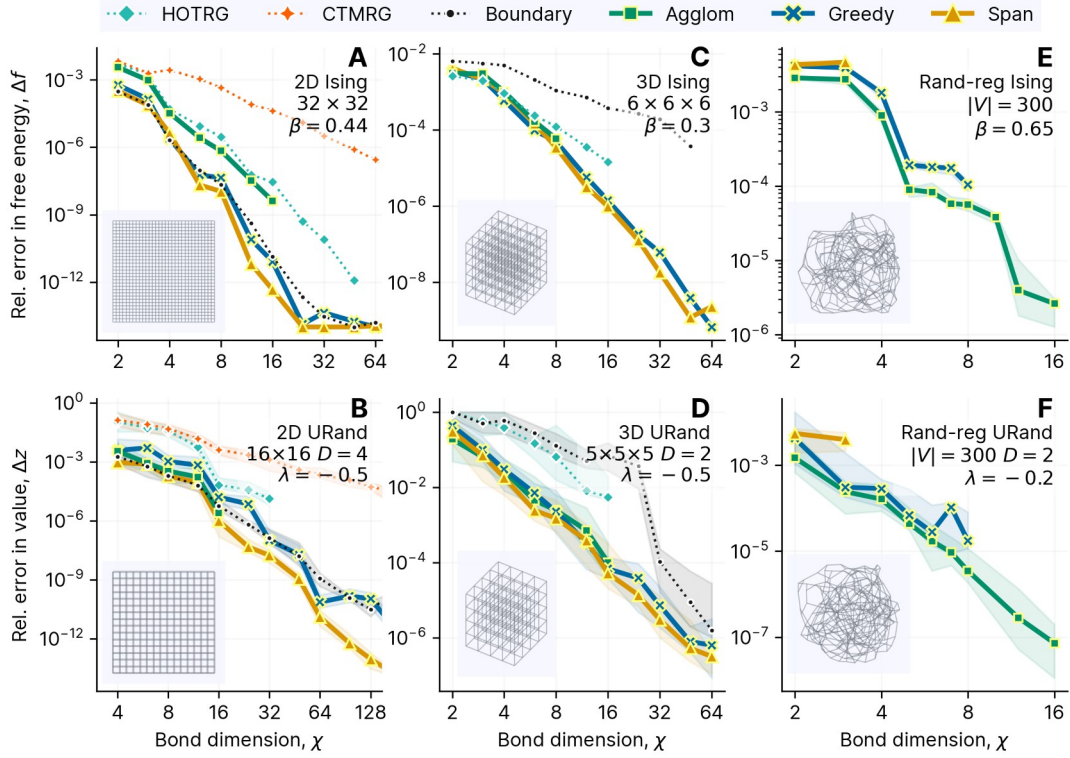


FIG. 11. Error versus compressed bond dimension χ of hyper-optimized approximate contraction using optimized Span, Greedy, and Agglom trees, in comparison to boundary contraction, CTMRG, and HOTRG, for medium size TNs. (Insets show sample geometry). In terms of gauging settings for the hyper-optimized 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).

ror. In Figs. 8B and 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 hyper-optimized Greedy or B&B approximate contraction trees is indeed much lower than that of boundary contraction. Interestingly, the heavily optimized ‘B&B’ tree does not improve on the error of the Greedy tree for a given peak memory M .

III. BENCHMARKING HYPER-OPTIMIZED 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 TRG

(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. We define a 3D version of (PEPS) boundary contraction on a cube that first contracts from one face of the cube towards 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 in-homogeneous systems. Illustrations of all the algorithms are given in the SI [57].

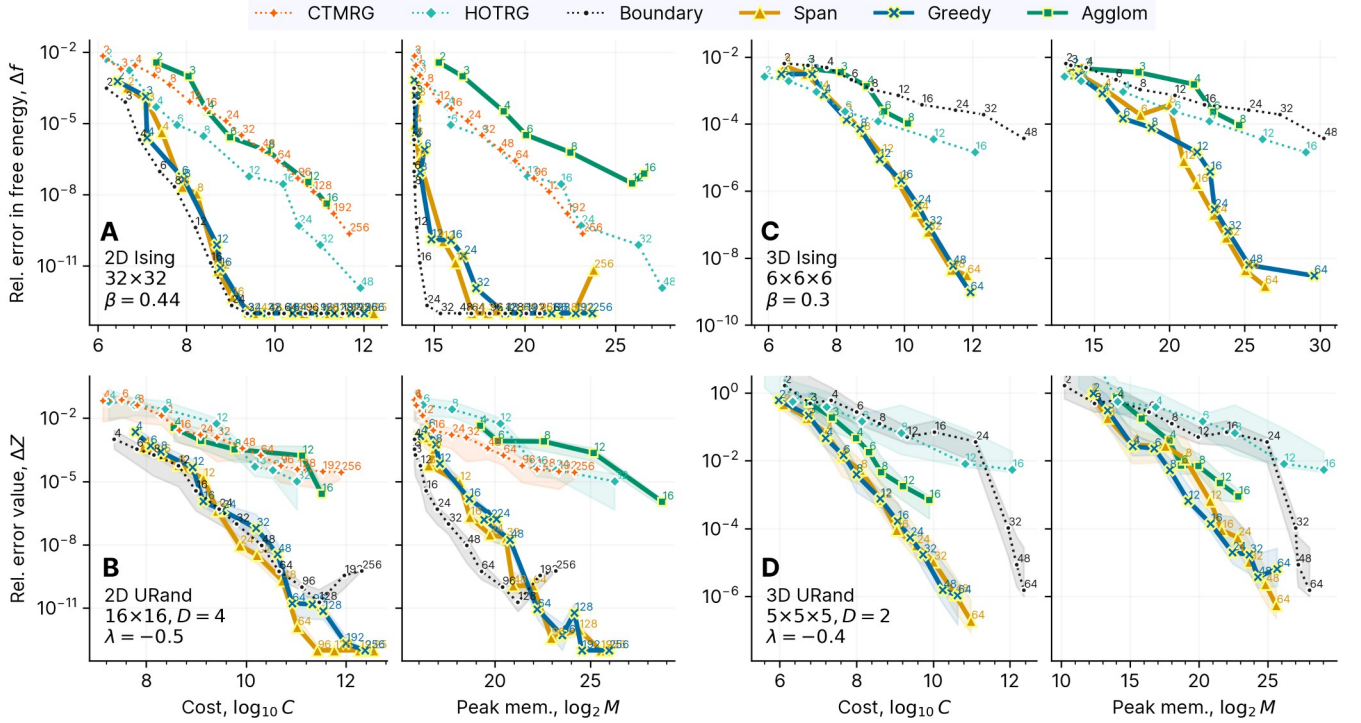


FIG. 12. Error versus cost of hyper-optimized approximate contraction using optimized Span, Greedy, and Agglom trees, in comparison to boundary contraction, CTMRG, and HOTRG, for medium size TNs where the exact reference values are available. Here the error is plotted against either the total cost of 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 α . The gauging settings used for the hyper-optimized methods are $r = 6$ and late compression in 2D and $r = 2$ and early compression in 3D.

B. Cost scaling with graph size

In Fig. 9 we show the computational cost (memory and floating point cost) of hyper-optimized 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 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 hyper-optimized Span tree performs a boundary-like contraction, but rather than contracting from one face across to the other side, it can find a strategy that contracts all faces towards 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 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 boundary-like 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.

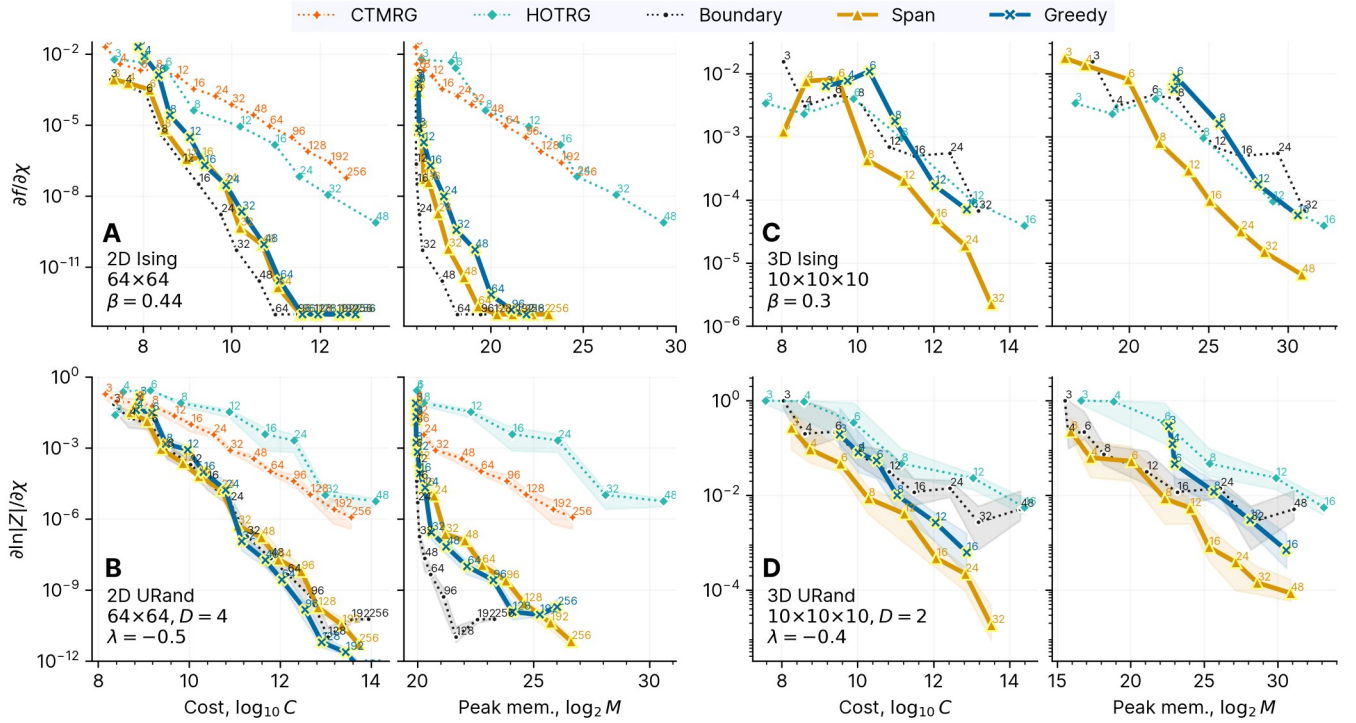


FIG. 13. Error versus cost of hyper-optimized approximate contraction using optimized Span, Greedy, and Agglom trees, in comparison to boundary contraction, CTMRG, and HOTRG, for large TNs where no exact reference is available. As a proxy for error, we monitor the rate of change of the log contraction 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 hyper-optimized methods use gauging settings of $r = 6$ in 2D and $r = 2$ in 3D, both with early compression.

C. Error versus bond dimension

As discussed above, we optimize over the generated contraction trees for a given bond dimension χ . In Fig. 11, we plot the relative error in the contraction value ΔZ /free energy per site Δf for 2D and 3D Ising and random tensor models for the hyper-optimized 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 $\chi \rightarrow \infty$ 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 hyper-optimized 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 hyper-optimized strategies for given χ . For fixed bond dimension, the hyper-optimized 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 hyper-optimized 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 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/memory is considered, because this depends on additional details besides χ , such as

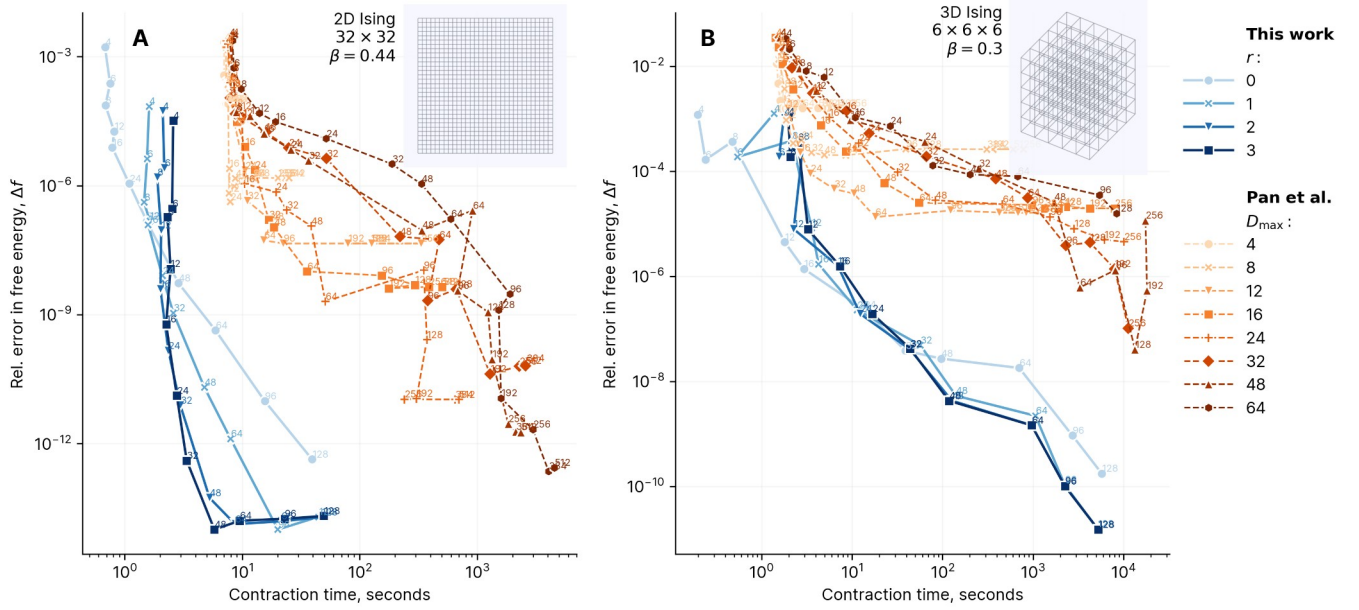


FIG. 14. Performance comparison of hyper-optimized approximate contraction (current work) and the algorithm of Pan et al. [52] for computing the free energy of the Ising model at approximately the critical point of **A**: a square lattice and **B**: a cubic lattice. For both algorithms χ is varied and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

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, hyper-optimized 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 hyper-optimized approximate contraction strategy to another recently proposed technique for arbitrary graphs. Ref. [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 hyper-optimized Span trees for the 2D/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 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 hyper-optimized contraction since one generally re-uses this many times. However as a rough guide, for the lattices in Fig. 14 the search converges to a good tree in 10–20 seconds. Further details and comparisons are in the SI [57]. We see clearly that in both the 2D and 3D cases (Figs. 14A and B respectively) the hyper-optimized 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. THE POWER OF HYPER-OPTIMIZED APPROXIMATE CONTRACTION

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

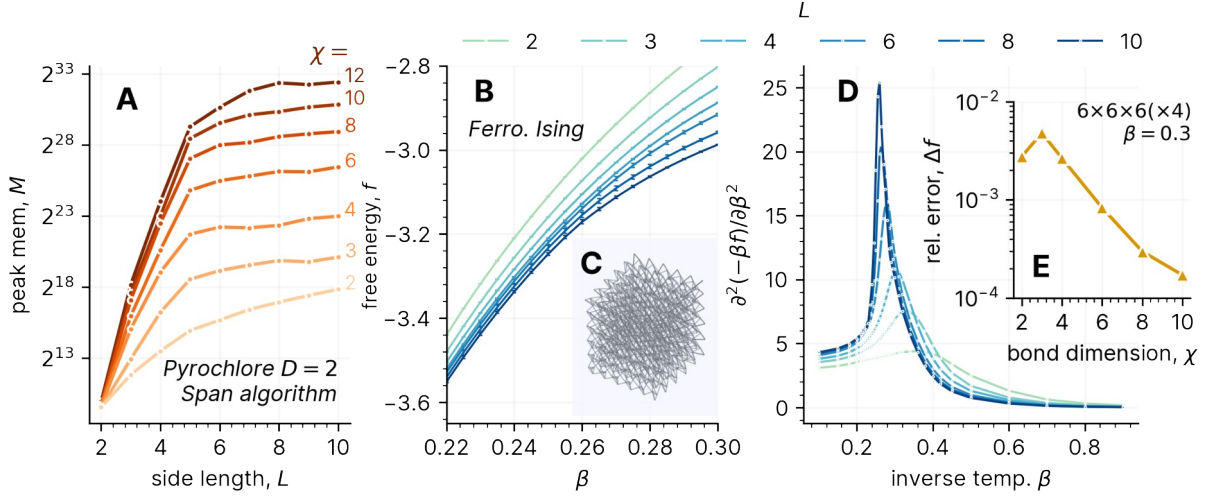


FIG. 15. Approximate contraction of the ferromagnetic Ising model on the pyrochlore lattice. **A**: peak memory M , of the Span algorithm on the pyrochlore lattice with $D=2$ as function of side length L and χ . **B**: the free energy, f , near the critical point, estimated by extrapolating approximate 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 $r=2$.

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. 15A 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. 15B, 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]. Fig. 15D shows the second derivative of $\frac{1}{N} \ln Z$ 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. 15C. For this size we can investigate the free energy error of the approximate contraction scheme, Δf , and this is shown in Fig. 15E. 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. 16A. Here, because the length of loops in the 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. 16C). To study the accuracy we consider the problem of counting dimer coverings on these graphs. This is equivalent to so-called positive #1-IN-3SAT [36, 74, 75]. Each edge (i.e. index) is considered a potential dimer, and by placing the tensor

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

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. 16B. 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, $1/\chi$,

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

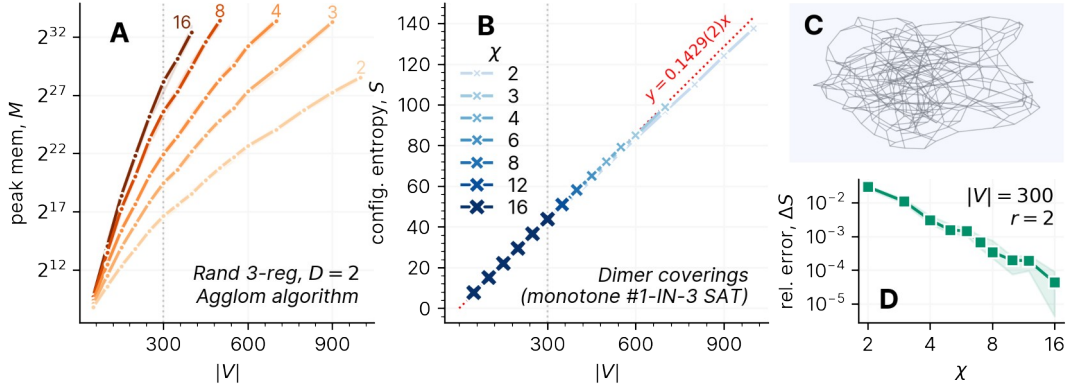


FIG. 16. Approximate contraction of dimer covering counting on random 3-regular graphs. Quantities are averaged over 20 instances. **A**: peak memory, M , of the Agglom algorithm on random 3-regular graph instances with $D = 2$ as a function of number of vertices, $|V|$. **B**: configuration entropy, $S = \ln W$, 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 $\ln|V|$ and χ (see main text). **C**: example random regular graph for $|V| = 300$. **D**: relative error in S as a function of χ for $|V| = 300$.

with fitted parameters, S_∞ , $C_{1...5}$, we can estimate the infinite size entropy per site as $S_\infty = 0.1429(2)$. The theoretical value of this can be computed using [78] when $|V| \gtrsim 5 \times 10^{11}$ (see the SI), yielding 0.1438, suggesting a small systematic error remains from the finite size. In Fig. 16D 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. 17A 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. 17B 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. 17C. 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 hyper-optimizing 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 hand crafted 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 “non-hard” 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

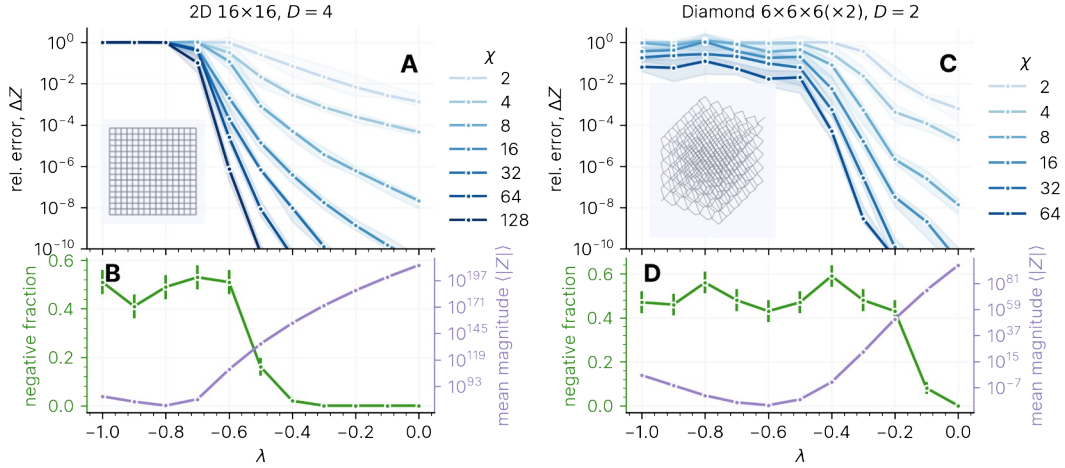


FIG. 17. Hardness transition in approximately contracting tensor networks with random uniform entries $\in [\lambda, 1]$. **A**: relative error, ΔZ , in approximately contracted value of the URand model on the square lattice using the Greedy algorithm as a function of λ and X with $r = 2$. Line and bands 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 axis) and average absolute magnitude (purple, right 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 X 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 axis) and average absolute magnitude (purple, right axis). Error bars denote error on mean.

scale exponentially, but with a usefully reduced pre-factor, 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 US National Science Foundation through Award No. 1931328. The approximate contraction algorithms were developed with support from the US National Science Foundation through Award No. 2102505. JG acknowledges support through a gift from Amazon Web Services Inc. GKC is partially supported as a Simons Investigator in Physics.

-
- [1] I. Markov and Y. Shi, “Simulating Quantum Computation by Contracting Tensor Networks,” *SIAM Journal on Computing*, vol. 38, pp. 963–981, Jan. 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 [quant-ph]*, Oct. 2017.
 - [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 [quant-ph]*, Dec. 2017.
 - [4] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, “Classical Simulation of Intermediate-Size Quantum Circuits,” *arXiv:1805.01450 [quant-ph]*, May 2018.
 - [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,” *arXiv:1811.09599 [quant-ph]*, Nov. 2018.
 - [6] J. Gray and S. Kourtis, “Hyper-optimized tensor network contraction,” *Quantum*, vol. 5, p. 410, Mar. 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 [quant-ph]*, May 2020.
 - [8] G. Kalachev, P. Panteleev, and M.-H. Yung, “Recursive Multi-Tensor Contraction for XEB Verification of Quantum Circuits,” *arXiv:2108.05665 [quant-ph]*, Aug. 2021.
 - [9] F. Pan and P. Zhang, “Simulating the Sycamore quantum supremacy circuits,” Mar. 2021.
 - [10] S. R. White, “Density-matrix algorithms for quantum renormalization groups,” *Physical Review B*, vol. 48, pp. 10345–10356, Oct. 1993.
 - [11] S. R. White, “Density matrix formulation for quantum renormalization groups,” *Physical Review Letters*, vol. 69, pp. 2863–2866, Nov. 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,” *Physical Review A*, vol. 75, p. 033605, Mar. 2007.
- [13] F. Verstraete and J. I. Cirac, “Renormalization algorithms for Quantum-Many Body Systems in two and higher dimensions,” *arXiv:cond-mat/0407066*, July 2004.
- [14] G. Vidal, “Classical Simulation of Infinite-Size Quantum Lattice Systems in One Spatial Dimension,” *Physical Review Letters*, vol. 98, p. 070201, Feb. 2007.
- [15] H. C. Jiang, Z. Y. Weng, and T. Xiang, “Accurate Determination of Tensor Network State of Quantum Lattice Models in Two Dimensions,” *Physical Review Letters*, vol. 101, p. 090603, Aug. 2008.
- [16] E. Stoudenmire and S. R. White, “Studying Two-Dimensional Systems with the Density Matrix Renormalization Group,” *Annual Review of Condensed Matter Physics*, vol. 3, no. 1, pp. 111–128, 2012.
- [17] P. C. G. Vlaar and P. Corboz, “Simulation of three-dimensional quantum systems with projected entangled-pair states,” *arXiv:2102.06715 [cond-mat, physics:quant-ph]*, Feb. 2021.
- [18] T. Nishino and K. Okunishi, “Corner Transfer Matrix Renormalization Group Method,” *Journal of the Physical Society of Japan*, vol. 65, pp. 891–894, Apr. 1996.
- [19] T. Nishino and K. Okunishi, “Corner Transfer Matrix Algorithm for Classical Renormalization Group,” *Journal of the Physical Society of Japan*, vol. 66, pp. 3040–3047, Oct. 1997.
- [20] M. Levin and C. P. Nave, “Tensor renormalization group approach to two-dimensional classical lattice models,” *Physical review letters*, vol. 99, no. 12, p. 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,” *Physical Review B*, vol. 80, p. 094403, Sept. 2009.
- [22] Z. Y. Xie, H. C. Jiang, Q. N. Chen, Z. Y. Weng, and T. Xiang, “Second Renormalization of Tensor-Network States,” *Physical Review Letters*, vol. 103, p. 160601, Oct. 2009.
- [23] L. Vanderstraeten, B. Vanhecke, and F. Verstraete, “Residual entropies for three-dimensional frustrated spin systems with tensor networks,” *Physical Review E*, vol. 98, p. 042145, Oct. 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,” *Physical Review B*, vol. 93, no. 12, p. 125115, 2016.
- [25] J.-G. Liu, L. Wang, and P. Zhang, “Tropical tensor network for ground states of spin glasses,” *Physical Review Letters*, vol. 126, no. 9, p. 090506, 2021.
- [26] A. J. Ferris and D. Poulin, “Tensor networks and quantum error correction,” *Phys. Rev. Lett.*, vol. 113, p. 030501, Jul 2014.
- [27] S. Bravyi, M. Suchara, and A. Vargo, “Efficient algorithms for maximum likelihood decoding in the surface code,” *Phys. Rev. A*, vol. 90, p. 032326, Sep 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*, vol. 9, p. 041031, Nov 2019.
- [29] C. T. Chubb and S. T. Flammia, “Statistical mechanical models for quantum codes with correlated noise,” *Annales de l’Institut Henri Poincaré D*, vol. 8, no. 2, pp. 269–321, 2021.
- [30] C. T. Chubb, “General tensor network decoding of 2d pauli codes,” *arXiv preprint arXiv:2101.04125*, 2021.
- [31] J. P. Bonilla Ataides, D. K. Tuckett, S. D. Bartlett, S. T. Flammia, and B. J. Brown, “The xxx surface code,” *Nature communications*, vol. 12, no. 1, pp. 1–12, 2021.
- [32] T. Farrelly, R. J. Harris, N. A. McMahon, and T. M. Stace, “Tensor-Network Codes,” *Physical Review Letters*, vol. 127, p. 040507, July 2021.
- [33] N. Schuch, M. M. Wolf, F. Verstraete, and J. I. Cirac, “Computational complexity of projected entangled pair states,” *Physical review letters*, vol. 98, no. 14, p. 140506, 2007.
- [34] A. García-Sáez and J. I. Latorre, “An exact tensor network for the 3sat problem,” *arXiv preprint arXiv:1105.3201*, 2011.
- [35] J. D. Biamonte, J. Morton, and J. Turner, “Tensor Network Contractions for #SAT,” *Journal of Statistical Physics*, vol. 160, pp. 1389–1404, Sept. 2015.
- [36] S. Kourtis, C. Chamon, E. Mucciolo, and A. Ruckenstein, “Fast counting with tensor networks,” *SciPost Physics*, vol. 7, no. 5, p. 060, 2019.
- [37] J. M. Dudek, L. Duenas-Orsorio, and M. Y. Vardi, “Efficient contraction of large tensor networks for weighted model counting through graph decompositions,” *arXiv preprint arXiv:1908.04381*, 2019.
- [38] J. M. Dudek and M. Y. Vardi, “Parallel weighted model counting with tensor networks,” *arXiv preprint arXiv:2006.15512*, 2020.
- [39] N. de Beaudrap, A. Kissinger, and K. Meichanetzidis, “Tensor network rewriting strategies for satisfiability and counting,” *arXiv preprint arXiv:2004.06455*, 2020.
- [40] A. J. Gallego and R. Orus, “Language design as information renormalization,” *arXiv preprint arXiv:1708.01525*, 2017.
- [41] V. Pestun and Y. Vlassopoulos, “Tensor network language model,” *arXiv preprint arXiv:1710.10248*, 2017.
- [42] T.-D. Bradley, E. M. Stoudenmire, and J. Terilla, “Modeling sequences with quantum states: a look under the hood,” *Machine Learning: Science and Technology*, vol. 1, no. 3, p. 035008, 2020.
- [43] K. Meichanetzidis, A. Toumi, G. de Felice, and B. Coecke, “Grammar-aware question-answering on quantum computers,” *arXiv preprint arXiv:2012.03756*, 2020.
- [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,” *Advances in physics*, vol. 57, no. 2, pp. 143–224, 2008.
- [45] R. Orús, “Tensor networks for complex quantum systems,” *Nature Reviews Physics*, vol. 1, no. 9, pp. 538–550, 2019.
- [46] B. Dittrich, F. C. Eckert, and M. Martin-Benito, “Coarse graining methods for spin net and spin foam models,” *New Journal of Physics*, vol. 14, p. 035008, Mar. 2012.
- [47] G. Evenbly and G. Vidal, “Tensor Network Renormalization,” *Physical Review Letters*, vol. 115, p. 180405, Oct. 2015.
- [48] S. Yang, Z.-C. Gu, and X.-G. Wen, “Loop optimization for tensor network renormalization,” *Physical Review Letters*, vol. 118, p. 110504, Mar. 2017.
- [49] M. Bal, M. Mariën, J. Haegeman, and F. Verstraete, “Renormalization Group Flows of Hamiltonians Using Tensor Networks,” *Physical Review Letters*, vol. 118, p. 250602, June 2017.
- [50] S.-J. Ran, E. Tarrico, C. Peng, X. Chen, L. Tagliacozzo, G. Su, and M. Lewenstein, *Tensor network contractions: methods and applications to quantum many-body systems*. Springer Nature, 2020.
- [51] A. Jermyn, “Automatic contraction of unstructured tensor networks,” *SciPost Physics*, vol. 8, no. 1, p. 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.*, vol. 125, p. 060503, Aug 2020.
- [53] There is a minor effect on memory.
- [54] G. Evenbly and G. Vidal, “Tensor network renormalization,” *Physical review letters*, vol. 115, no. 18, p. 180405, 2015.

- [55] M. Hauru, C. Delcamp, and S. Mizera, “Renormalization of tensor networks using graph-independent local truncations,” *Physical Review B*, vol. 97, no. 4, p. 045111, 2018.
- [56] S. Yang, Z.-C. Gu, and X.-G. Wen, “Loop optimization for tensor network renormalization,” *Physical review letters*, vol. 118, no. 11, p. 110504, 2017.
- [57] J. Gray and G. K.-L. Chan, “Supplementary information for “hyper-optimized approximate contraction of tensor networks with arbitrary geometry,” 2022.
- [58] G. Evenbly, “Gauge fixing, canonical forms, and optimal truncations in tensor networks with closed loops,” *Physical Review B*, vol. 98, no. 8, p. 085155, 2018.
- [59] M. Hauru, C. Delcamp, and S. Mizera, “Renormalization of tensor networks using graph independent local truncations,” *Physical Review B*, vol. 97, p. 045111, Jan. 2018.
- [60] L. Wang and F. Verstraete, “Cluster update for tensor network states,” Oct. 2011.
- [61] P. Corboz, T. M. Rice, and M. Troyer, “Competing states in the t-J model: Uniform d-wave state versus stripe state,” *Physical Review Letters*, vol. 113, p. 046402, July 2014.
- [62] S. Iino, S. Morita, and N. Kawashima, “Boundary Tensor Renormalization Group,” *Physical Review B*, vol. 100, p. 035449, July 2019.
- [63] S. Schlag, *High-Quality Hypergraph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020.
- [64] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, “High-quality hypergraph partitioning,” *ACM J. Exp. Algorithmics*, mar 2022.
- [65] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- [66] J. Rapin and O. Teytaud, “Nevergrad - A gradient-free optimization platform.” <https://GitHub.com/FacebookResearch/Nevergrad>, 2018.
- [67] T. Nishino and K. Okunishi, “Corner transfer matrix renormalization group method,” *Journal of the Physical Society of Japan*, vol. 65, no. 4, pp. 891–894, 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,” *Physical Review B*, vol. 86, no. 4, p. 045139, 2012.
- [69] S. Bramwell and M. Harris, “Frustration in ising-type spin models on the pyrochlore lattice,” *Journal of Physics: Condensed Matter*, vol. 10, no. 14, p. 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 A: Statistical Mechanics and its Applications*, vol. 450, pp. 541–545, May 2016.
- [71] K. Soldatov, K. Nefedev, Y. Komura, and Y. Okabe, “Large-scale calculation of ferromagnetic spin systems on the pyrochlore lattice,” *Physics Letters A*, vol. 381, pp. 707–712, Feb. 2017.
- [72] F. Wang and D. P. Landau, “Efficient, Multiple-Range Random Walk Algorithm to Calculate the Density of States,” *Physical Review Letters*, vol. 86, pp. 2050–2053, Mar. 2001.
- [73] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of State Calculations by Fast Computing Machines,” *The Journal of Chemical Physics*, vol. 21, pp. 1087–1092, June 1953.
- [74] J. Raymond, A. Sportiello, and L. Zdeborová, “Phase diagram of the 1-in-3 satisfiability problem,” *Physical Review E*, vol. 76, p. 011101, July 2007.
- [75] L. Zdeborová and M. Mézard, “Constraint satisfaction problems with isolated solutions are hard,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, p. P12004, Dec. 2008.
- [76] T. J. Schaefer, “The complexity of satisfiability problems,” in *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC ’78, (New York, NY, USA), pp. 216–226, Association for Computing Machinery, May 1978.
- [77] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1979.
- [78] B. Bollobás and B. D. McKay, “The number of matchings in random regular graphs and bipartite graphs,” *Journal of Combinatorial Theory, Series B*, vol. 41, no. 1, pp. 80–91, 1986.
- [79] T. Grover and M. P. A. Fisher, “Entanglement and the sign structure of quantum states,” *Physical Review A*, vol. 92, p. 042308, Oct. 2015.
- [80] L. A. Goldberg and H. Guo, “The complexity of approximating complex-valued Ising and Tutte partition functions,” Jan. 2017.
- [81] A. Galanis, L. A. Goldberg, and A. Herrera-Poyatos, “The complexity of approximating the complex-valued Ising model on bounded degree graphs,” May 2021.
- [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 of Mathematics, Sigma*, vol. 10, 2022/ed.
- [83] M. A. Beauchamp, “An improved index of centrality,” *Behavioral science*, vol. 10, no. 2, pp. 161–163, 1965.
- [84] M. Marchiori and V. Latora, “Harmony in the small-world,” *Physica A: Statistical Mechanics and its Applications*, vol. 285, no. 3–4, pp. 539–546, 2000.
- [85] L. Onsager, “Crystal statistics. i. a two-dimensional model with an order-disorder transition,” *Physical Review*, vol. 65, no. 3–4, p. 117, 1944.
- [86] R. J. Baxter, *Exactly solved models in statistical mechanics*. Elsevier, 2016.
- [87] F. Pan, P. Zhou, S. Li, and P. Zhang, “CATN.” <https://github.com/panzhang83/catn>, 2019.
- [88] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, pp. 357–362, Sept. 2020.

Hyper-optimized approximate contraction of tensor networks with arbitrary geometry: supplementary information

CONTENTS

I. Introduction	1
II. A Framework for Approximate Contraction Algorithms	2
A. Components of approximate contraction	2
B. Models for testing	3
C. Bond compression strategies	4
D. Early versus late compression	5
E. Comparison of the tree gauge to other gauges	6
F. Approximate contraction algorithm	6
G. Generating contraction trees	7
H. Optimizing the contraction trees	8
I. Quality of hyper-optimization	9
III. Benchmarking hyper-optimized approximate contraction trees	10
A. Summary of hand-coded strategies for regular lattices	10
B. Cost scaling with graph size	11
C. Error versus bond dimension	12
D. Error versus cost	12
E. Comparison to another strategy for general graphs	13
IV. The power of hyper-optimized approximate contraction	13
A. Ising partition function on the pyrochlore lattice	14
B. Random 3-regular graphs and dimer coverings	14
C. Hardness transition in random tensor networks	15
V. Conclusions	15
Acknowledgments	16
References	16
A. Tree spans and gauging	20
B. Explicit projector form	20
C. Tree builder details	22
1. Greedy	23
2. Span	23
3. Agglom	24
4. Branch & bound approximate contraction tree	24
D. Tree cost functions	24
E. Hand-coded Contraction Schemes	25
F. Models	26
1. Ising Model	26
2. URand Model	29
3. Dimer covers / positive #1-IN-3SAT	29
G. Performance comparison to CATN	31
H. Performance on corner double line (CDL) tensor networks	32

Appendix A: Tree spans and gauging

In this section we detail the simple method of generating a (possibly r -local) spanning tree, for use with the tree gauge, and also in the `Span` contraction tree building algorithm (note the spanning tree is a different object to the contraction tree). A pseudocode outline is given in Algorithm. 2. We first take an arbitrary initial connected subgraph, S_0 of the graph, G , for example the two tensors sharing bonds that are about to be compressed. We then greedily select a pair of nodes, one inside and one outside this region, which expands the spanning tree, τ and region S , until all nodes within graph distance r of S_0 are in S . Since the nodes can share multiple edges, the spanning tree τ is best described using an ordered set of node pairs rather than edges. If the graph G has cycles, then nodes outside S may have multiple connections to it, this degeneracy is broken by choosing a scoring function.

Algorithm 2 r -local spanning tree

Input: graph G, initial region S_0, max distance r $\tau \leftarrow \emptyset$ $S \leftarrow S_0$ $c \leftarrow \emptyset$ for $u \in S$ do for $v \in \text{NEIGHBORS}(G, u) \setminus S$ do $r_{uv} \leftarrow 1$ $c \leftarrow c \cup \{(u, v, r_{uv})\}$ end for end for while $c > 0$ do $u, v, r_{uv} \leftarrow \text{BEST}(c)$ $c \leftarrow c \setminus \{(u, v, r_{uv})\}$ if $(v \notin S) \wedge (r_{uv} \leq r)$ then $S \leftarrow S \cup \{v\}$ $\tau \leftarrow \tau \cup \{(u, v)\}$ for $w \in \text{NEIGHBORS}(G, v) \setminus S$ do $r_{vw} \leftarrow r_{uv} + 1$ $c \leftarrow c \cup \{(v, w, r_{vw})\}$ end for end if end while Return: τ, S	$\triangleright$ ordered set of pairs forming spanning tree $\triangleright$ set of nodes spanned by the tree $\triangleright$ candidates to add to tree $\triangleright$ each node in original region $\triangleright$ connected nodes not in region $\triangleright$ distance to original region $\triangleright$ pop the best candidate edge $\triangleright$ node is new and close enough $\triangleright$ add v to region $\triangleright$ add edge to tree $\triangleright$ add new neighboring candidates
---	--

For the tree gauge, we choose the scoring function such that the closest node with the highest connectivity (product of sizes of connecting edge dimensions) is preferred. The gauging proceeds by taking the pairs in τ in reverse order, gauging bonds from the outer to the inner tensor (see main text Fig.3E). If the graph G is a tree and we take $r = \infty$, this corresponds exactly to canonicalization of the region S_0 . In order to perform a compression of a bond in the tree gauge, we just need to perform QR decompositions inwards on a ‘virtual copy’ of the tree (as shown in the main text Fig. 4B), until we have the central ‘reduced factors’ R_A and R_B . Performing a truncated SVD on the contraction of these two to yield $R_A R_B = R_{AB} \approx U, \sigma, V^\dagger$, allows us to compute the locally optimal projectors to insert on the bond as $P_L = R_B V \sigma^{-1/2}$ and $P_R = \sigma^{-1/2} U^\dagger R_A$ such that $AB \approx AP_L P_R B$. The form of these projectors, which is the same as CTMRG and HOTRG but including information up to distance r away, is explicitly derived in Sec. B. One further restriction we place is to exclude any tensors from the span that are input rather than intermediate tensors.

One obvious alternative possibility to the tree-gauge is to introduce an initial ‘Simple Update’ style gauge [15] on each of the bonds and update these after compressing a bond, including in the vicinity of the adjacent tensors. A similar scheme was employed for 3D contractions in [17]. In our experience this performs similarly to the tree-gauge (and indeed the underlying operations are very similar) but is more susceptible to numerical issues due to the direct inversion of potentially small singular values.

Appendix B: Explicit projector form

Performing a bond compression such as in the main text Fig. 3D can be equated to the insertion of two approximate projectors that truncate the target bond to size χ . The projector form allows us to perform the tree gauge compression ‘virtually’ - i.e. without having to modify tensors anywhere else in the original tensor network. We begin by considering the product AB , where A and B might represent collections of tensors such as a local tree. Assuming we can decompose each into an orthogonal and ‘reduced’

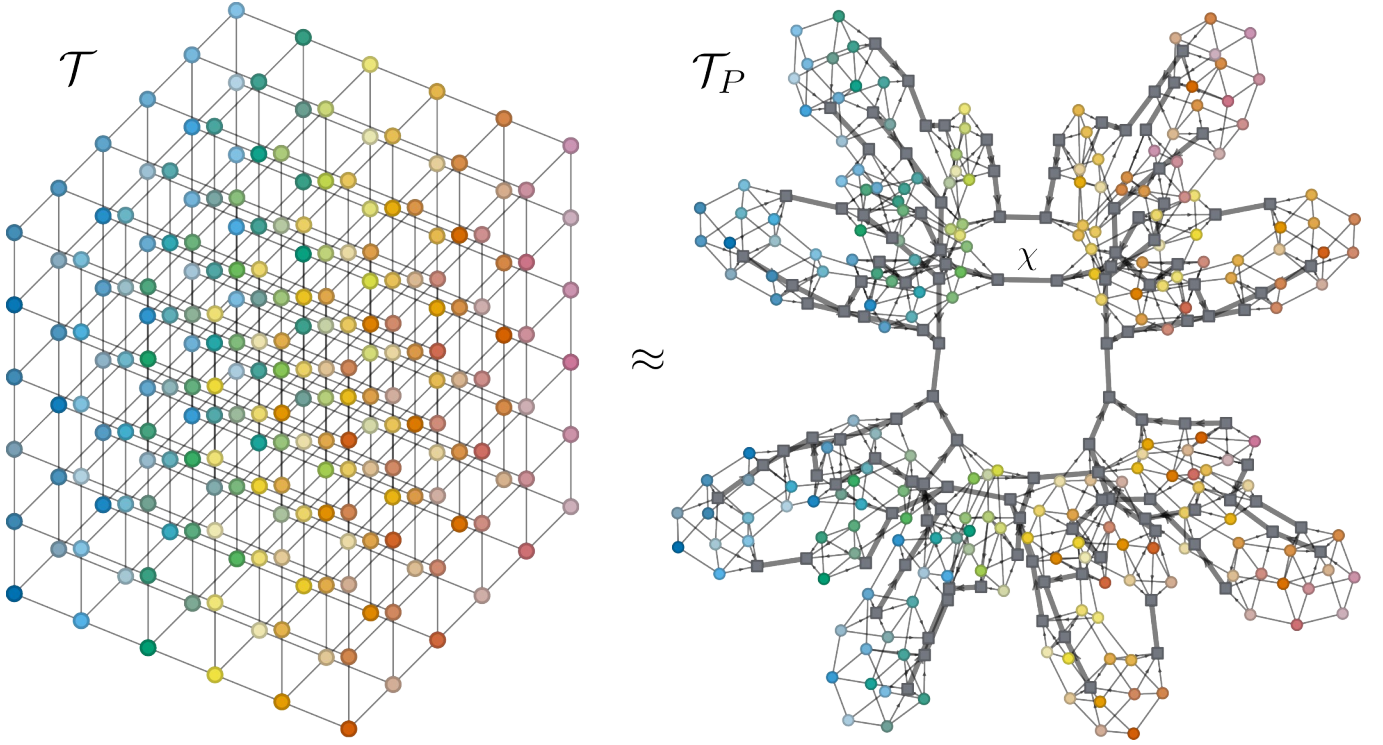


FIG. 18. An example of transforming a tensor network, $\mathcal{T}$, into an exactly contractable tensor network, $\mathcal{T}_P$, using the explicit projector form of a approximate contraction tree. Here we take a $6 \times 6 \times 6$ with $D = 2$ cube and use an optimized contraction tree from the `Span` generator, for $\chi = 8$. Each node in the original tensor network is colored uniquely. The grey square nodes in the right hand side diagram represent the inserted projectors, with thicker edges the compressed bonds of size χ . Arrows indicate the orientation of the projectors (i.e. the order of the compressions).

factor we write:

$$AB = (Q_A R_A)(R_B Q_B).$$

If we resolve the identity on either side, we can form the product $R_A R_B$ in the middle and perform a truncated SVD on this combined reduced factor yielding $U \sigma V^\dagger$.

$$\begin{aligned} AB &= Q_A R_A (R_A^{-1} R_A) (R_B R_B^{-1}) R_B Q_B \\ &= Q_A R_A R_A^{-1} (U \sigma V^\dagger) R_B^{-1} R_B Q_B \\ &= Q_A R_A (R_A^{-1} U \sqrt{\sigma}) (\sqrt{\sigma} V^\dagger R_B^{-1}) R_B Q_B \end{aligned}$$

from which we can read off the projectors that we need to insert into the original tensor network in order to realize the optimal truncation as:

$$\begin{aligned} P_L &= R_A^{-1} U \sqrt{\sigma}, \\ P_R &= \sqrt{\sigma} V^\dagger R_B^{-1}. \end{aligned}$$

Finally, in order to avoid performing the inversion of the reduced factors, we can simplify:

$$\begin{aligned} P_L &= R_A^{-1} U \sqrt{\sigma} \\ &= R_A^{-1} (U \sqrt{\sigma} \sqrt{\sigma} V^\dagger) V \sigma^{-1/2} \\ &= R_A^{-1} (R_A R_B) V \sigma^{-1/2} \\ &= R_B V \sigma^{-1/2} \end{aligned}$$

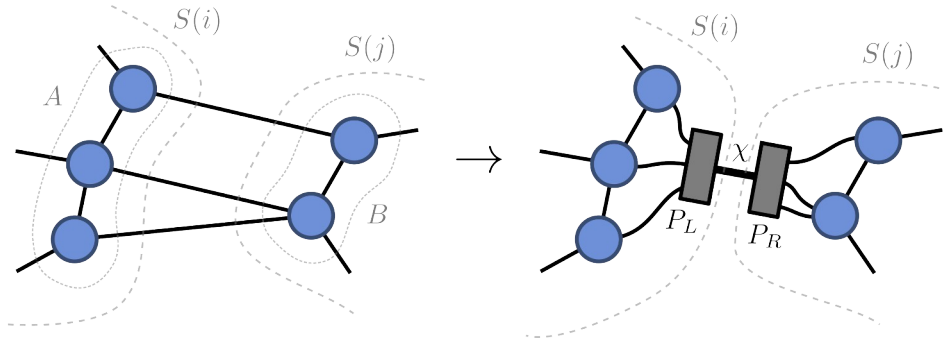
and likewise:

$$\begin{aligned}
 P_R &= \sqrt{\sigma} V^\dagger R_B^{-1} \\
 &= \sigma^{-1/2} U^\dagger (U \sqrt{\sigma} \sqrt{\sigma} V^\dagger) R_B^{-1} \\
 &= \sigma^{-1/2} U^\dagger (R_A R_B) R_B^{-1} \\
 &= \sigma^{-1/2} U^\dagger R_A.
 \end{aligned}$$

This form of the projectors makes explicit the equivalence to CTMRG and HOTRG [60–62], for which R_A and R_B contain only information about the local plaquette. Note in general that we just need to know R_A and R_B (not Q_A or Q_B) to compute P_L and P_R , but we can include in these the effects of the distance- ℓ tree gauge in order to perform the truncation locally without modifying any tensors but A and B .

Rather than dynamically performing the approximate contraction algorithm using the ordered contraction tree, one can also use it to statically map the original tensor network, T , to another tensor network, T_P , which has the sequence of projectors lazily inserted into it (i.e. each $A P_L P_R B$ is left uncontracted). *Exact* contraction of T_P then gives the approximate contracted value of T . Such a mapping may be useful for relating the approximate contraction to other tensor network forms [50], or for performing some operations such as optimization [22]. Here we describe the process.

To understand where the projectors should be inserted we just need to consider the sub-graphs that the intermediate tensors correspond to. At the beginning of the contraction, each node corresponds to a sub-graph of size 1, containing only itself. We can define the sub-graph map $S(i) = \{i\}$ for $i = 1 \dots N$. When we contract two nodes i, j to form a new node k , the new sub-graph is simply $S(k) = S(i) \cup S(j)$. When we compress between two intermediate tensors i and j , we find all bonds connecting $S(i)$ to $S(j)$, and insert the projectors P_L and P_R , effectively replacing the identity linking the two regions with the rank- χ operator $P_L P_R$. Finally we add the tensor P_L to the sub-graph $S(i)$ and P_R to the sub-graph $S(j)$. This can be visualized like so.



Grouping all the neighboring tensors on one side of the bonds as an effective matrix A and those on the other side as B (note that these might generally include projectors from previous steps), the form of P_L and P_R can be computed as above.

An example of the overall geometry change of performing this explicit projection transformation for the full set of compressions on a cubic tensor network approximate contraction is shown in Fig. 18. Note that the dynamic nature of the projectors, which depend on both the input tensors and the contraction tree, is what differentiates a tensor network which you contract using approximate contraction, and for instance directly using a tree- or fractal-like ansatz such as T_P .

Appendix C: Tree builder details

In this section we provide extended details of each of the heuristic ordered contraction tree generators. First we outline the hyper optimization approach. Each tree builder B takes as input the graph G with edges weighted according to the tensor network bond sizes, as well as a set of heuristic hyper-parameters, θ , that control how it generates an ordered contraction tree Y . The builder is run inside a hyper-optimization loop that uses a generic optimizer, O , to sample and tune the parameters. We use the *nevergrad* [66] optimizer for this purpose. A scoring function computes some metric $\mathcal{Y}$ for each tree (see Sec. D for possible functions), which is used to train the optimizer and track the best score and tree sampled so far, $\mathcal{Y}_{best}$ and Y_{best} respectively. The result, outlined in Algorithm 3, is an anytime algorithm (i.e. can be terminated at any point) that samples trees from a space that progressively improves. Note that while the optimization targets a specific χ , the tree produced exists separately from χ and can be used for a range of values of χ (in which case one would likely optimize for the maximum value).

Algorithm 3 Hyper optimization loop**Input:** graph G , max bond X , builder B , optimizer O $Y_{best} \leftarrow \infty$ **while** *optimizing* **do** $\theta \leftarrow \text{SAMPLE PARAMETERS } (O)$ $Y \leftarrow \text{GENERATE TREE } (B, G, \theta)$ $Y \leftarrow \text{SCORE TREE } (Y, X)$ **if** $Y < Y_{best}$ **then** $Y_{best} \leftarrow Y$ $Y_{best} \leftarrow Y$ **end if** $\text{REPORT PARAMETERS } (O, \theta, Y)$ **end while****Return:** Y_{best}

$\triangleright$ Get new hyper parameters
 $\triangleright$ Build tree with new parameters
 $\triangleright$ Score the tree

$\triangleright$ Update optimizer with score

In the following subsections we outline the specific hyper parameter choices, $\bar{\theta}$, for each tree builder. However one useful recurring quantity is a measure of *centrality*, similar to the harmonic closeness[83, 84], that assigns to each node a value according to how central it is in the network. This can be computed very efficiently as $c^v = \frac{1}{Z} \sum_{u \neq v} \frac{\sqrt{1}}{d(u,v)+1}$, where $d(u, v)$ is the shortest distance between nodes u and v . The normalization constant Z is chosen such that $c^v \in [0, 1] \forall v$.

1. Greedy

The Greedy algorithm builds an ordered contraction tree by taking the graph at step α of the contraction, G_α , and greedily selecting a pair of tensors to contract $(i, j) \rightarrow k$, simulating the contraction and compression of those tensors, and then repeating the process with the newly updated graph, $G_{\alpha+1}$, until only a single tensor remains. The pair of tensors chosen at each step are those that minimize a local scoring function, and it is the parameters within this that are hyper-optimized. The local score is a sum of the following components:

- $\log_2$ size of new tensor *after* compression with weight $\theta_{new\ size}$.
- $\log_2$ size of new tensor *before* compression with weight $\theta_{old\ size}$.
- The minimum, maximum, sum, mean or difference (the choice of which is a hyper parameter) of the two input tensor sizes $\log_2$, with weight θ_{inputs} .
- The minimum, maximum, sum, mean or difference (the choice of which is a hyper parameter) of the sub-graph sizes of each input (when viewed as sub-trees) with weight $\theta_{subgraph}$.
- The minimum, maximum, mean or difference (the choice of which is a hyper parameter) of the centralities of each input tensor with weight $\theta_{centrality}$. Centrality is propagated to newly contracted nodes as the minimum, maximum or average of inputs (the choice of which is a hyper-parameter).
- a random variable sampled from the Gumbel distribution multiplied by a temperature (which is a hyper-parameter).

The final hyper-parameter is a value of X_{greedy} to simulate the contraction with, which can thus deviate from the real value of X used to finally score the tree. The overall space defined is 11-dimensional, which is small enough to be tuned by, for example, Bayesian optimization. In our experience it is not crucial to understand how each hyper-parameter affects the tree generated, other than that they are each chosen to carry some meaningful information from which the optimizer can conjure a local contraction strategy; the approach is more in the spirit of high-dimensional learning rather than a physics-inspired optimization.

2. Span

The Span algorithm builds an ordered contraction tree using a modified, tunable version of the spanning tree generator in Algorithm 2 with $r = \infty$. The basic idea is to interpret the ordered sequence of node pairs in the spanning tree, τ , as the reversed series of contractions to perform. The initial region S_0 is taken as one of the nodes with the highest or lowest centrality (the choice being a hyper-parameter). The remaining hyper-parameters are used to tune the local scoring function (BEST (c) in Algorithm. 2), that decides which pair of nodes should be added to the tree at each step. These are:

- The connectivity of the candidate node to the current region, with weight $\theta_{connectivity}$.

- The dimensionality of the candidate tensor, with weight θ_{ndim} .
- The distance of the candidate node from the initial region, with weight $\theta_{distance}$
- The centrality of the candidate node, with weight $\theta_{centrality}$
- a random variable sampled from the Gumbel distribution multiplied by a temperature (which is a hyper-parameter).

The final hyper-parameter is a permutation controlling which of these scores to prioritize over others.

3. Agglom

The `Agglom` algorithm builds the contraction tree by repeated graph partitioning using the library `KaHyPar` [63, 64]. We first partition the graph, G into $\sim |V|/K$ parts, with the target subgraph size K being a tunable hyper-parameter. Another hyper-parameter is the imbalance, $\theta_{imbalance}$, which controls how much the sub-graph sizes are allowed to deviate from K . Other hyper-parameters at this stage pertain to `KaHyPar`:

- θ_{mode} either ‘direct’ or ‘recursive’,
- $\theta_{objective}$ either ‘cut’ or ‘km1’,
- θ_{weight} whether to weight the edges constantly or logarithmically according to bond size.

Once a partition has been formed, the graph is transformed by simulating contracting all of the tensors in each group, and then compressing between the new intermediates to create a new graph with $\sim |V|/K$ nodes and bonds of size no more than X_{agglom} (itself a hyper-parameter which can deviate from the real X used to score the tree). The contractions within each partition are chosen according to the `Greedy` algorithm. Finally, the tree generated in this way is not ordered. To fix an ordering the contractions are sorted by sub-graph size and average centrality.

4. Branch & bound approximate contraction tree

The hyper-optimized approach produces heavily optimized trees but with no guarantee that they are an optimal solution. For small graphs a depth first branch and bound approach can be used to find an optimal tree exhaustively, or to refine an existing tree if terminated early. The general idea is to run the greedy algorithm whilst tracking a score, but keep and explore every candidate contraction at each step (a ‘branch’) in order to ‘rewind’ and improve it. The depth first aspect refers to prioritizing exploring branches to completion so as to establish an upper bound on the score. The upper bound can then be improved and used to terminate bad branches early.

Appendix D: Tree cost functions

There are various cost functions one can assign to an approximate contraction tree to then optimize against. Broadly these correspond to either space (memory) or time (FLOPs) estimates. Three cost functions that we have considered that only depend on the tree and X (but not gauging scheme for example) are the estimated peak memory, M , the largest intermediate tensor, W , and the number of FLOPs involved in the contractions only. Specifically, given the set of tensors, $\{V_\alpha\}$, present at stage α of the contraction, the peak memory is given by:

$$M = \max_{\alpha} \sum_{v \in \{V_\alpha\}} \text{size}(T^v). \quad (D1)$$

Given a compression and gauging scheme, one can also trace through the full computation, yielding a more accurate peak memory usage, $\tilde{M}$, as well as an estimate of the FLOPs associated with all QR and SVD decompositions too – we call this the full computational ‘cost’, C . Included in this we consider only the dominant contributions:

- contraction of two tensors with effective dimensions (m, n) and (n, k) : mnk
- QR of tensor with effective dimensions (m, n) with $m \geq n$: $2mn^2 - \frac{2}{3}n^3$
- SVD of tensor with effective dimensions (m, n) with $m \geq n$: $4mn^2 - \frac{4}{3}n^3$.

Algorithm 4 Branch and bound tree search

```

Input: graph  $G$ , Maximum bond dimension  $\chi$ 
 $y_{best} \leftarrow \infty$ 
 $c = \{\}$ 
for  $i, j \in \text{EDGES}(G)$  do
     $y \leftarrow 0$ 
     $p \leftarrow []$ 
     $c \leftarrow c \cup \{(i, j, G, y, p)\}$ 
end for
while  $|c| > 0$  do
     $(i, j, G, y, p) \leftarrow \text{REMOVE\_BEST}(c)$ 
    if  $\text{INVALID}(i, j, G)$  or  $y \geq y_{best}$  then
        continue
    end if
    if  $|G| = 1$  and  $y < y_{best}$  then
         $y_{best} \leftarrow y$ 
         $p_{best} \leftarrow p$ 
        continue
    end if
     $p \leftarrow \text{APPEND}(p, (i, j))$ 
     $(k, G, y) \leftarrow \text{SIMULATE CONTRACTION}(i, j, G, \chi)$ 
    for  $l \in \text{NEIGHBORS}(G, k)$  do
         $c \leftarrow c \cup \{(k, l, G, y, p)\}$ 
    end for
end while
 $Y_{best} \leftarrow \text{BUILD TREE FROM PATH}(G, p_{best})$ 
Return:  $Y_{best}$ 

```

$\triangleright$ candidate contractions
 $\triangleright$ populate with every pair of tensors
 $\triangleright$ initial score
 $\triangleright$ the contraction ‘path’

 $\triangleright$ no need to explore further

 $\triangleright$ finished contraction with best score

 $\triangleright$ continue exploring
 $\triangleright$ k is the new node
 $\triangleright$ add new possible contractions

Of these the first two dominate since the SVD is only ever performed on the reduced bond matrix. Note the actual FLOPs will be a constant factor higher depending on the data type of the tensors.

In Fig. 19 we plot the relationship between the various metrics mentioned above for several thousand randomly sampled contraction trees on both a square and cubic geometry for varying D, χ and algorithm. We note that $M, \tilde{M}$ and W are all tightly correlated. The full cost C is slightly less correlated with these and only slightly more so with the ‘contractions only’ cost. Importantly however, the best contractions largely appear to simultaneously minimize all the metrics.

Appendix E: Hand-coded Contraction Schemes

In Figs. 20-24 we illustrate the various hand-coded contraction schemes used as comparisons in the text: 2D boundary contraction, 2D corner transfer matrix RG [67], 2D higher-order TRG [68], 3D PEPS boundary contraction, and 3D higher-order TRG [68]. Note that in the case of CTMRG and HOTRG, the algorithms are usually iterated to treat infinite, translationally invariant lattices, but here we simply apply a finite number of CTMRG or HOTRG steps and also generate the projectors locally to handle in-homogeneous tensor networks. For both CTMRG and HTORG we use the cheaper, ‘lazy’ method [62] of computing the reduced factors R_A and R_B which avoids needing to form and compute a QR on each pair of tensors on either side of a plaquette. We then use the projector form as given in Sec. B to compress the plaquette. The 3D PEPS boundary contraction algorithm has not previously been implemented to our knowledge, but is formulated in a way analogous to 2D boundary contraction. Notably, if any dimension is of size 1 it reduces to exactly 2D boundary contraction including canonicalization. For further details, we refer to the lecture notes [50] and the original references.

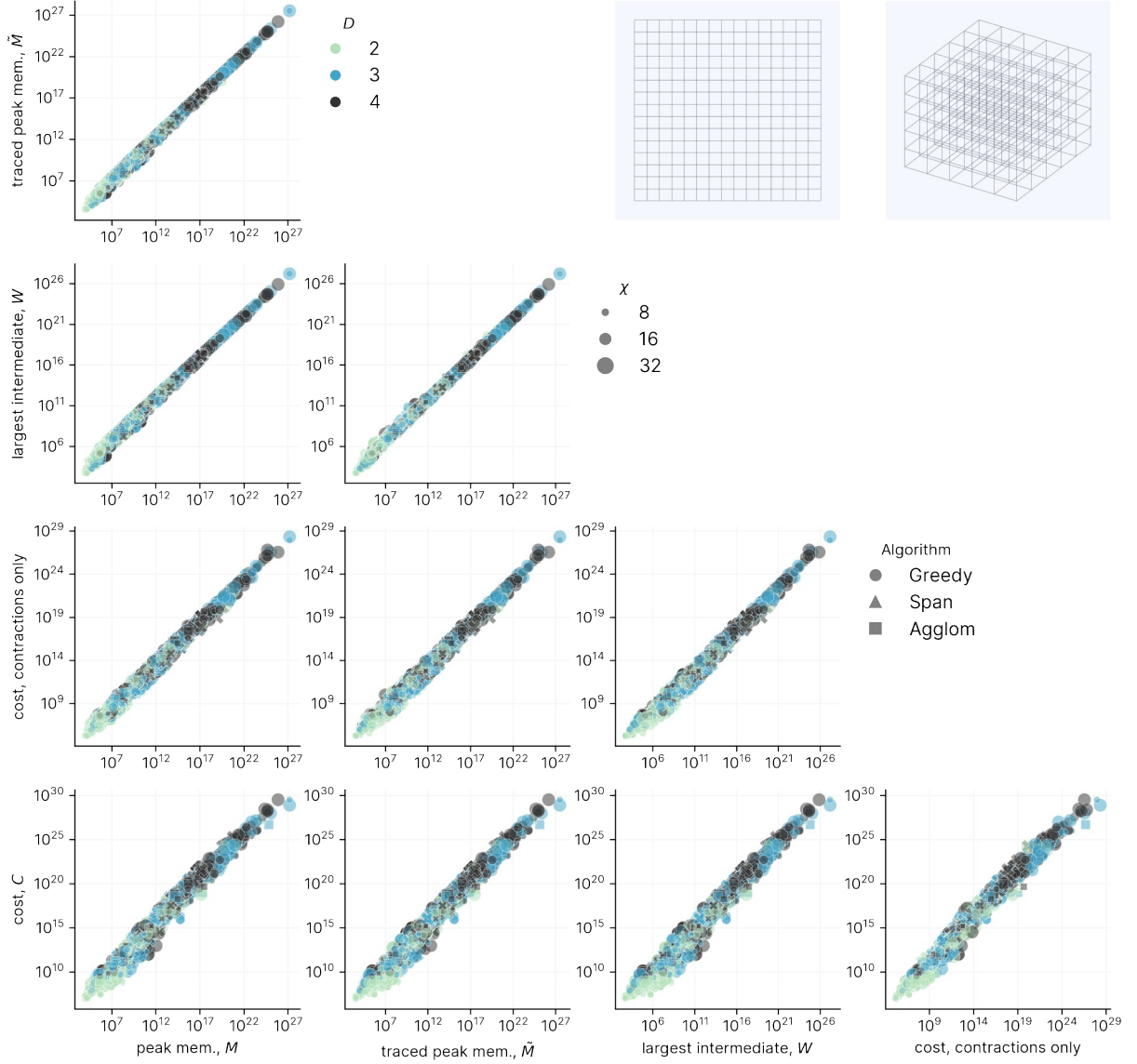


FIG. 19. Relationship between various tree cost functions for randomly sampled approximate contraction trees for two geometries: 2D square of size 16×16 and 3D cube of size $6 \times 6 \times 6$ (pictured in insets). D , X , the algorithm and its hyper-parameters are all uniformly sampled.

Appendix F: Models

1. Ising Model

We consider computing the free energy per spin of a system of N classical spins at inverse temperature β ,

$$f = \frac{-\log Z}{N\beta} \quad (\text{F1})$$

where the partition function, Z , is given by:

$$Z = \sum_{\{\sigma\}} \prod_{\langle i,j \rangle} \exp(j\beta \sigma_i \sigma_j), \quad (\text{F2})$$

$\sigma_k \in [1, -1]$ being the state of spin k and $\{\sigma\}$ the set of all configurations. The interaction pairs $\langle i, j \rangle$ are the edges of the graph, G , under study. We take the interaction strength j to be 1, i.e. ferromagnetic. While Monte Carlo methods can readily compute

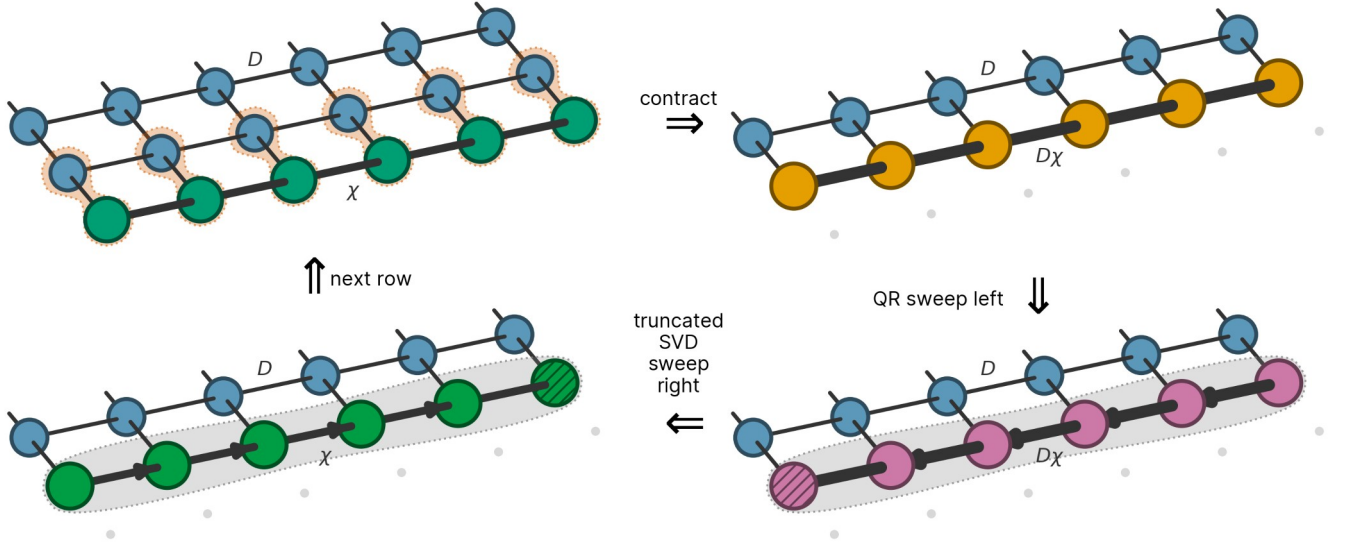


FIG. 20. Overview of a single step of the the manual 2D boundary contraction method that uses an MPS to sweep across the square.

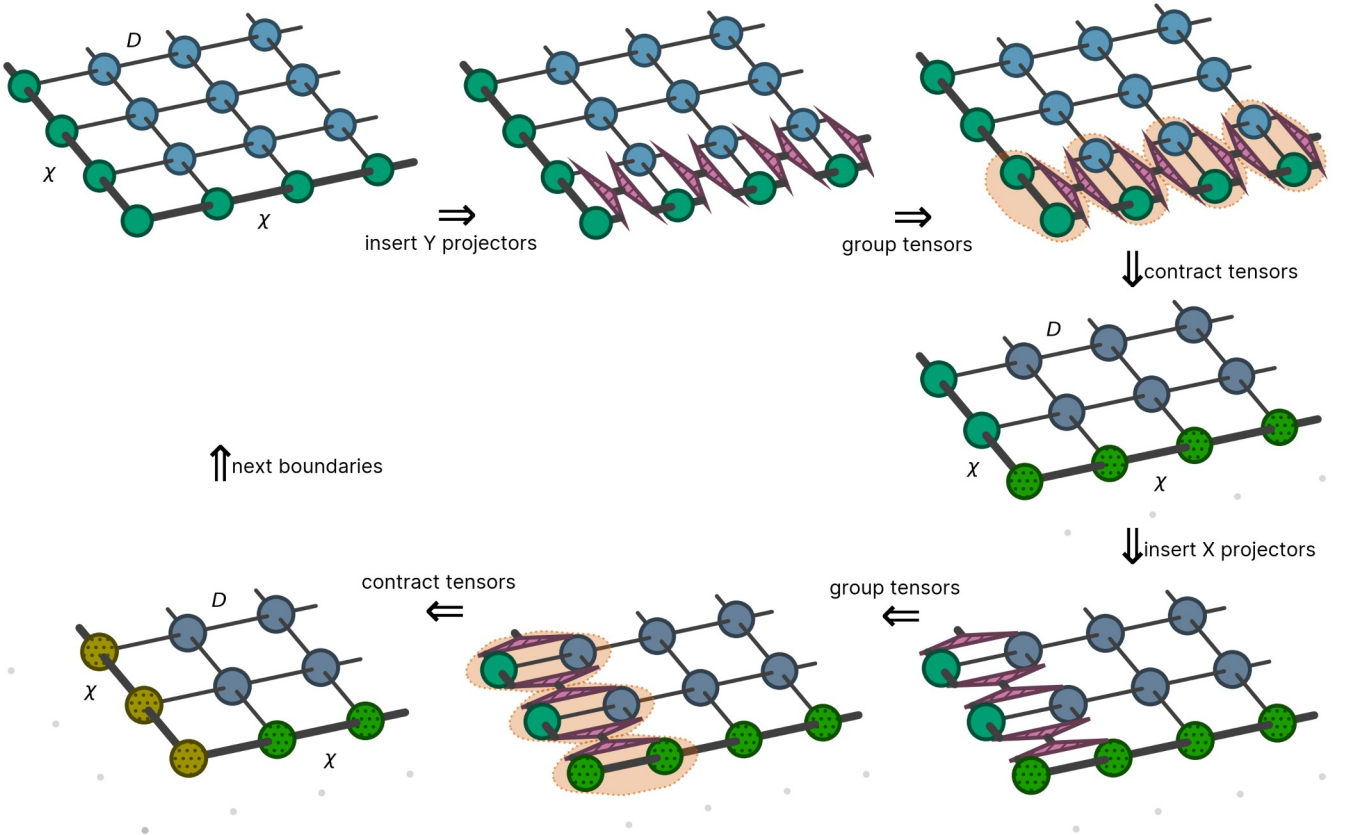


FIG. 21. Illustration of two boundary contraction steps of CTMRG for a finite 2D lattice. The full algorithm proceeds to contract all four of the sides inwards in succession. Note that the projectors (pink) are not identical across the lattice but are computed specific to the local tensors to allow for finite in-homogeneous systems.

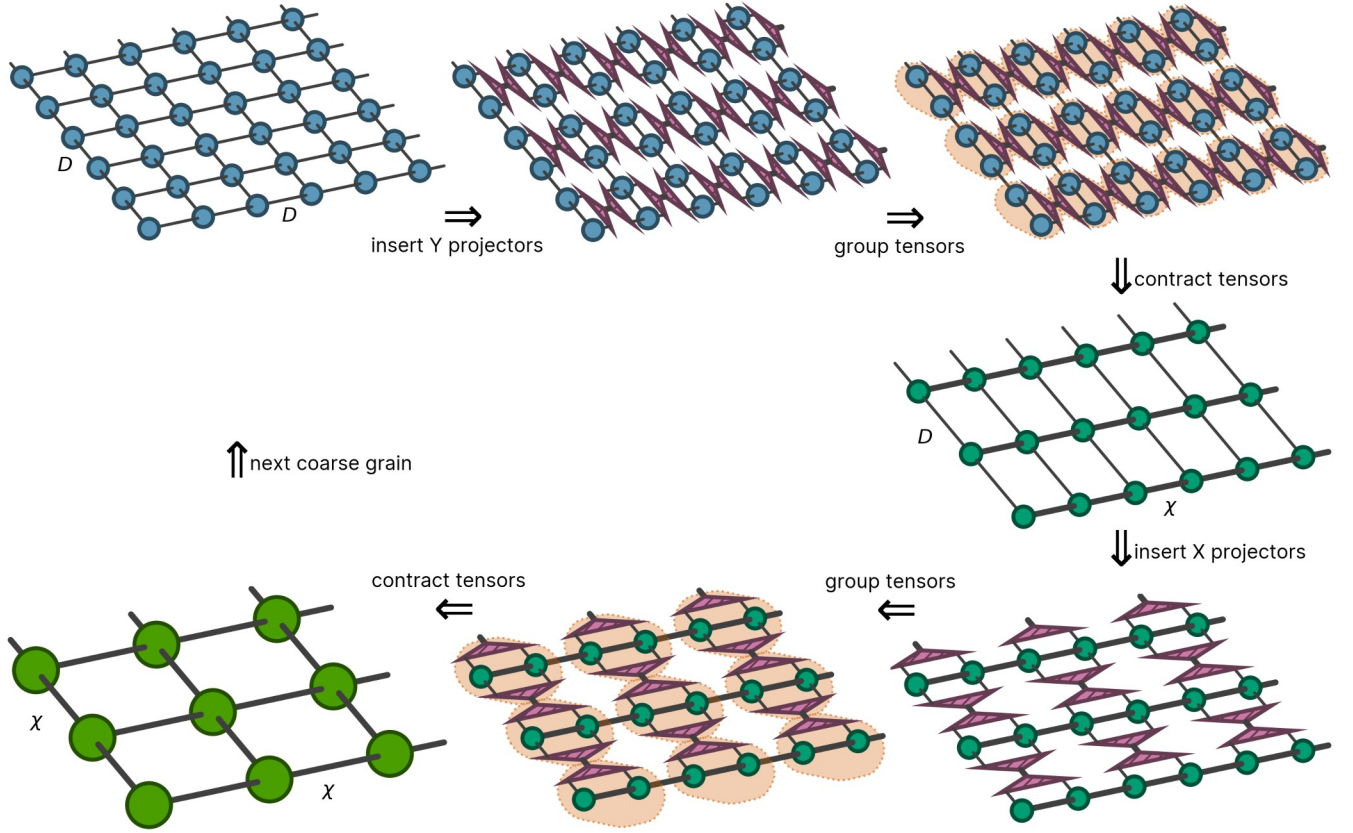


FIG. 22. Illustration of a full coarse graining step of HOTRG for a finite 2D lattice. Note that once a round of coarse graining has taken place, all bonds are of size X and so the next round starts with $D = X$. Note also that the projectors (pink) are not identical across the lattice but are computed specific to the local tensors to allow for finite in-homogeneous systems.

many quantities in such models, we note that the partition function and free energy are typically much more challenging [72]. Regardless of geometry we assume the spins are orientated in the same direction - the uniaxial Ising model. Typically one converts Z into a ‘standard’ tensor network with a single tensor per spin (or equivalently vertex of G), by placing the tensor,

$$T_{\{e_v\}}^{[v]} = \prod_{i \in \{e_v\}} \prod_{e_j \in \{e_v\}}^X \prod_{e_l \in \{e_v\}}^Y W_{i,e_j} \quad (F3)$$

on each vertex v of G , where the matrix W is defined by $(W^2)_{ij} = \exp(\beta \sigma_i \sigma_j)$. For $j > 0$ we can define W as real and symmetric using:

$$W = \frac{1}{2} \begin{pmatrix} \frac{p}{\cosh(j\beta)+} & \frac{p}{\sinh(j\beta)} \\ \frac{p}{\cosh(j\beta)-} & \frac{p}{\sinh(j\beta)} \end{pmatrix} \begin{pmatrix} \frac{p}{\cosh(j\beta)-} & \frac{p}{\sinh(j\beta)} \\ \frac{p}{\cosh(j\beta)+} & \frac{p}{\sinh(j\beta)} \end{pmatrix}.$$

This is equivalent to splitting the matrix on each bond then contracting each factor into a COPY-tensor placed on each vertex. We note that while this yields a tensor network with the exact geometry of the interaction graph G , one could factorize the COPY-tensor in other low-rank ways. Indeed for the exact reference results the TN in Eq.(F2) is contracted directly by interpreting every spin state index as a hyper index (i.e. appearing on an arbitrary number of tensors). The relative error in the free energy is given by:

$$\Delta f = 1 - \frac{f}{f_{\text{exact}}} = 1 - \frac{\log Z}{\log Z_{\text{exact}}} \quad (F4)$$

with f_{exact} results obtained via exact contraction. Depending on geometry the Ising model undergoes a phase transition at critical temperature β_c in the thermodynamic limit and it is in this vicinity that generally Δf peaks for finite systems. For example, on the 2D square lattice the exact value is known, $\beta_c = \frac{\log(1+\sqrt{2})}{2} \approx 0.44$ [85, 86].

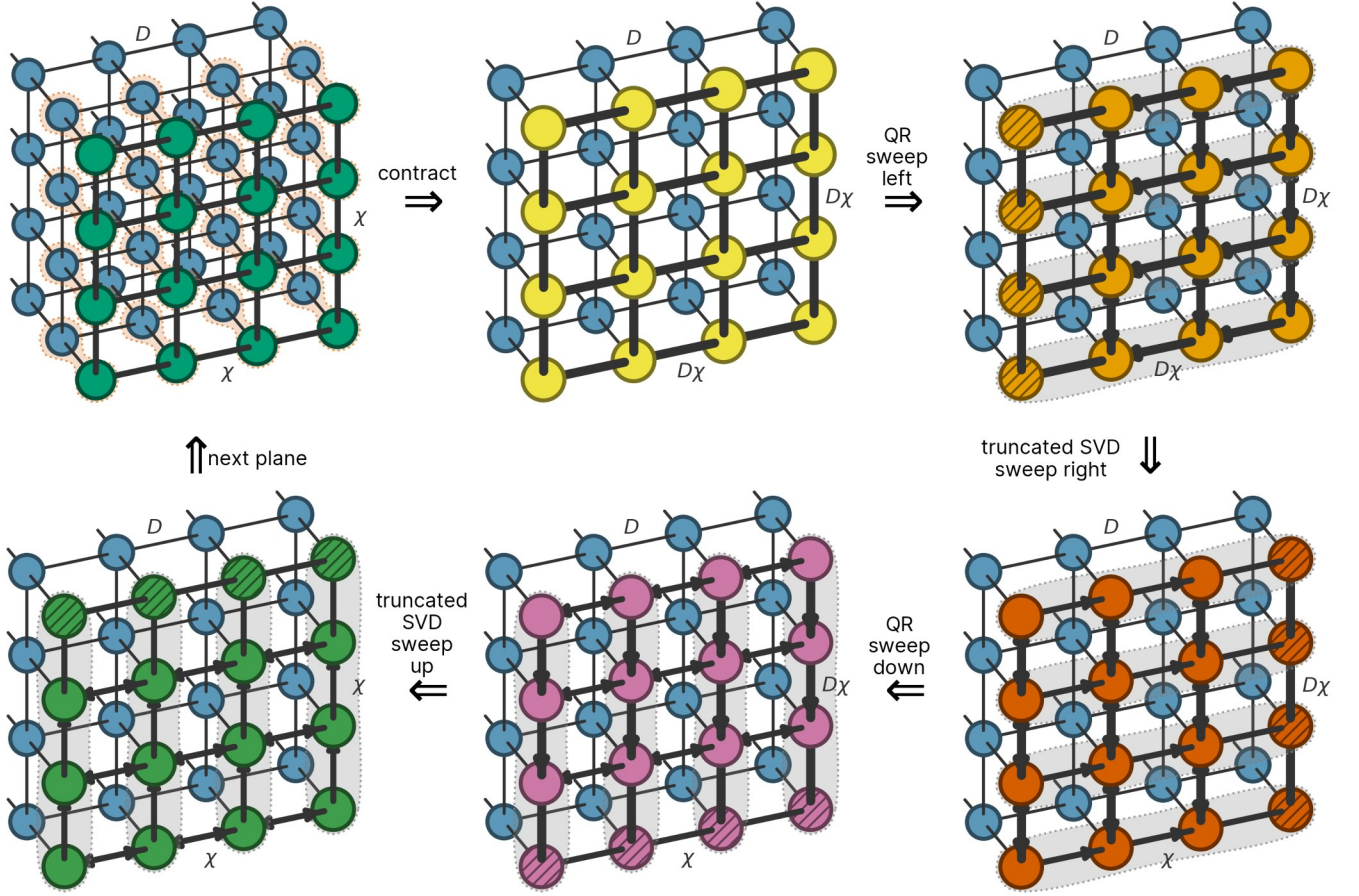


FIG. 23. Illustration of a single step of the manual 3D boundary contraction method that uses a PEPS to sweep across the cube. When L_x , L_y or $L_z = 1$ the scheme becomes equivalent to MPS boundary contraction.

2. URand Model

While the Ising model varies in difficulty depending on β , it seems always relatively easy to approximate to some extent using approximate contraction. On the other hand we expect there to be tensor networks which are exponentially difficult to approximate even for simple geometries. Here we introduce the *URand* model which allows us to continuously tune between very hard and very easy regimes. This is achieved simply by filling each tensor with random values sampled uniformly from the range $[\lambda, 1]$. When $\lambda \geq 0$, every term in the TN sum is non-negative and the sum becomes very easy to approximate. As λ becomes more negative however, the sum increasingly becomes terms of opposite sign which ‘destructively interfere’ making the overall contracted value Z hard to approximate. Choosing an intermediate λ allows us to generate ‘moderately hard’ contractions where the different gauging and tree generating strategies have a significant effect. For the URand model we consider the relative error in Z directly:

$$\Delta Z = 1 - \frac{Z}{Z_{\text{exact}}}$$

with Z_{exact} computed via exact contraction.

3. Dimer covers / positive #1-IN-3SAT

In this model we want to compute the entropy per site of dimer coverings of a graph G with number of vertices $|V|$. Here, a valid configuration is given if every vertex of G is ‘covered’ by exactly one dimer. Counting all valid configurations is done by enumerating every combination of placing a dimer on a bond (setting the corresponding index to 1) or not (setting the index to 0),

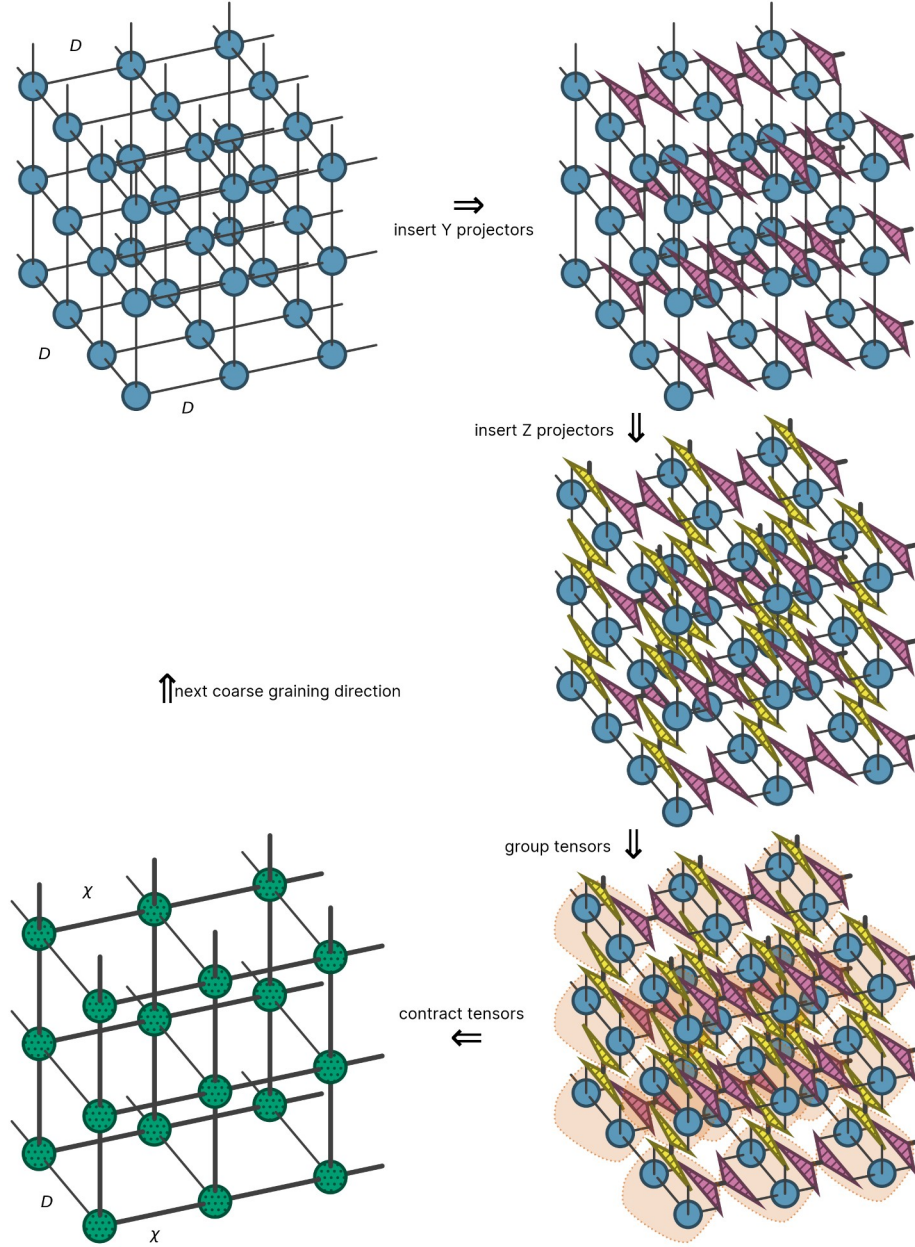


FIG. 24. Illustration of a single coarse graining step of HOTRG for a finite 3D lattice. For brevity we only show coarse graining in the x-direction but the full algorithm coarse grains each of the three dimensions in succession. Note that once two or more directions have been coarse grained, all bonds will be of size X and so subsequent rounds start with $D = X$. Note also that the projectors (pink and yellow) are not identical across the lattice but are computed specific to the local tensors to allow for finite in-homogeneous systems.

which can be formed as a tensor network with the following tensor on each vertex:

$$T_{i,j,k,\dots} = \begin{cases} 1, & \text{if } i + j + k + \dots = 1 \\ 0, & \text{otherwise} \end{cases}. \quad (\text{F5})$$

The total number of valid configurations is then the contraction:

$$W = \sum_{\{e\}} \prod_v T_{\{e_v\}}^{[v]} \quad (\text{F6})$$

This is also equivalent to the counting problem positive #1-IN-3SAT [36, 74, 75], the decision version of which is NP-

Complete [76, 77]. For 3-regular random graphs, this is known to be close the hardest regime though just on the side of satisfiability, in terms of the density of variables (edges) to clauses (vertices) [74, 75]. The residual entropy per site is given by:

$$S = \frac{\log W}{n}, \quad (\text{F7})$$

for number of vertices $n = |V|$, with relative error:

$$\Delta S = 1 - \frac{S}{S_{\text{exact}}}. \quad (\text{F8})$$

The reference values S_{exact} are computed using exact contraction, which is feasible up to $n \sim 300$ for 3-regular random graphs.

The problem is also known as counting ‘perfect matchings’, ‘complete matchings’, or ‘1-factors’ and has been studied for random regular graphs in the large $|V|$ limit [78]. There it was shown that if the degree k satisfies $3 \leq k < \log(n)^{\frac{1}{3}}$ then the expected value of W across all random k -regular instances is

$$\bar{W} = \left(\sqrt{\frac{k-1}{2}} + O(n^{-\frac{2}{3}}) \right) e^{\frac{1}{4}((k-1)^{k-1}/k^{k-2})^{n/2}}. \quad (\text{F9})$$

If we take the limit of this we find:

$$\begin{aligned} S_{\infty} &= \lim_{n \rightarrow \infty} \log \bar{W} / n \\ &= \lim_{n \rightarrow \infty} \frac{1}{2}(k-1) \log(k-1) + \frac{1}{2}(2-k) \log(k) + O\left(\frac{1}{n}\right) \\ &= 0.1438410362258904 \dots \end{aligned}$$

The condition linking k and n requires $n \geq 5.3 \times 10^{11}$, the scale of which suggests that our estimate of 0.1429(2) might have some small systematic error remaining from finite size effects.

Appendix G: Performance comparison to CATN

So far where appropriate we have compared our method to manually specified contraction orders in 2D and 3D. In [52], an algorithm to automatically contract arbitrary geometry tensor networks was also developed which showed good performance across a range of graphs. For convenience here we refer to that algorithm as CATN. While the basic tensor operations are similar, in CATN an effective periodic MPS is used to contract the graph using SWAP operations to remove bonds. A major difference is also that in this work we optimize the pattern of contractions and compressions ahead of time for the specific geometry. A python implementation of [52] is available at [87] which we can use to compare against for free energies of graphical models. Taking that code as is, the most direct way to compare performance of these two approaches is simply wall time on a single core of a CPU, here an AMD EPYC 7742.

In Fig. 25 we show a more detailed comparison than the main text of CATN and this current work for the Ising model at approximately the critical point on 2D and 3D lattices, as a function of accuracy vs contraction time. For our algorithm here we use the `Span` tree builder and show a range of X with the tree gauge distance $r = 2$. One consideration is that in our approach the hyper-optimization step might run separately to the actual contraction, since it depends only on the geometry and the approximate contraction tree can be re-used for different tensor entries (e.g. sweeping β). Here we show both the pure contraction only time and also the time if one takes the hyper-optimization into account. We compare to the contraction time reported directly by CATN, and note that this includes computing (greedily) which edge to remove next on-the-fly. In CATN, there are two bond dimension parameters controlling the trade-off between accuracy and computational effort, D_{max} and X , and two main parameters controlling how to select the bonds to remove, `select` and `reverse`. The relationship between D_{max} and X and the error and computational time is not trivial so we sweep across both. For the main text we showed `select=0` and `reverse=1` but here we also show the other good combinations. CATN also has four other parameters, `node`, `corder`, `swapopt`, and `svdopt`. We find these generally have no systematic or significant effect on error time for these examples, but nonetheless for each point take the best performing combination in terms of $\min \Delta F \times \text{time}$. The core linear algebra operations in both algorithms are performed using the same version of `numpy` [88], and we take the best time out of three repeats.

In both the 2D and 3D cases, Figs. 25A and B respectively, we see that our algorithm achieves the best accuracy vs. contraction time trade-off across the range of values and settings considered, especially in the high accuracy regime. We note that once hyper-optimization times are taken into account, for some less high accuracies CATN can perform better. For this comparison, we are interested in the automated performance of each algorithm, and thus use the greedy ordering of CATN. However, it was noted in [52] that an explicitly specified ‘Zig-Zag’ order performs well for the square 2D lattice, suggesting that optimizing over strategies might be beneficial for that approximate contraction scheme also.

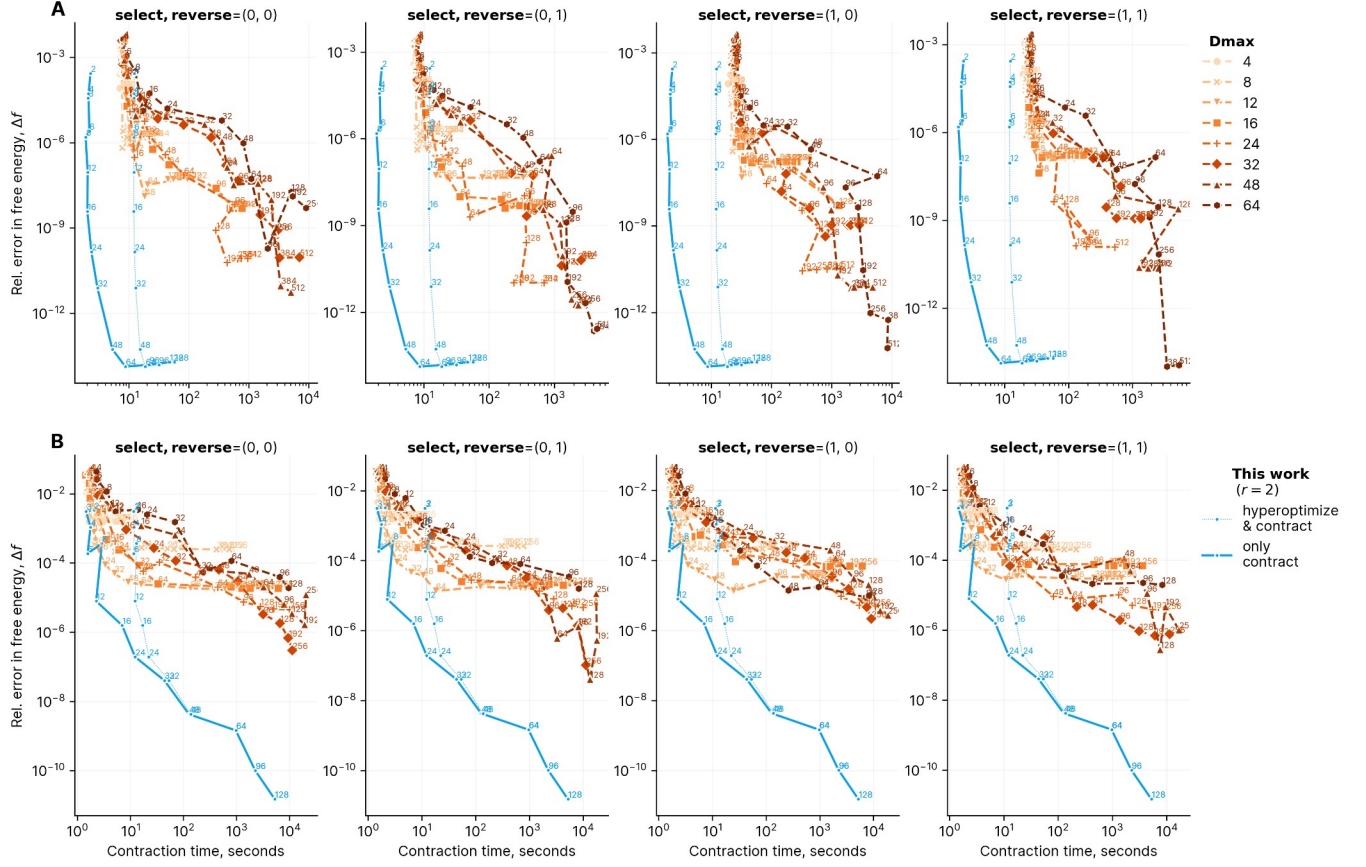


FIG. 25. Performance comparison of this current work (blue) and the algorithm of Pan et al. [52] (orange) using various settings for computing the free energy of the Ising model at approximately the critical point of **A**: a 32×32 square lattice at the approximate critical temperature $\beta=0.44$ and **B**: a $6 \times 6 \times 6$ cubic lattice at the approximate critical temperature $\beta=0.3$. For both algorithms X is varied and the points are labeled with the value. For the current work we show both the time with contraction only, and also accounting for the hyper-optimization time (about 10 seconds).

Appendix H: Performance on corner double line (CDL) tensor networks

An important model in the development of various normalization group style approximate contraction algorithms has been that of the corner double line (CDL) TN [47]. This model involves embedding local loop correlations in a lattice. Each such loop consists of four 2 dimensional COPY tensors (i.e. identity matrices) with dimension d placed around each *plaquette*. This results in four corner tensors at each *site* which can be contracted (via an outer product) to give a $d^2 \times d^2 \times d^2 \times d^2$ tensor after grouping indices. Each bond is thus doubled, carrying correlations from the two adjacent plaquettes. Such a CDL TN has a trivial, purely local correlation structure, that should not propagate to the coarse grained picture after a real space normalization procedure. However it is simple to show that both elementary algorithms such as TRG [20] and also more advanced algorithms such as HOTRG [68] never fully remove all such correlations, and it has been speculated that this is a source of error in approximate contraction schemes for more physically motivated models, which has sparked many improved schemes that explicitly handle the CDL correlations.

On the other hand, if one is only interested in the accuracy of the contracted value of the TN, then the CDL model poses no problem for all the contraction methods we consider here. Indeed the correlations are exactly the type that can be sustainably removed as the contraction proceeds, as long as X is above some very small threshold. In Fig. 26A we compare the relative error for contracting a 64×64 CDL TN with 5 methods and show that each becomes essentially exact at either $\chi = 2$ or $\chi = 4$ (for $d = 2$). Since we focus on open boundary conditions here, we use the CDL TN depicted in Fig. 26B with 0- and 1- dimensional COPY tensors along the boundary where necessary, however the same behavior holds for periodic boundary conditions. This is easily understood from the fact that once two pairs of tensors on either side of a plaquette have been contracted to A, B (which all these methods do) the internal plaquette correlation is ‘resolved’ into a scalar contribution, allowing the remaining local operator AB to be exactly represented with rank reduced by a factor of d^2 .

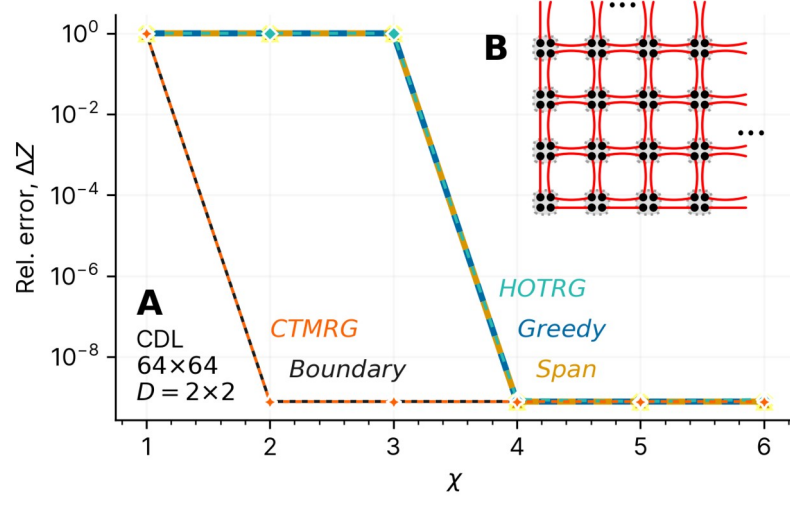


FIG. 26. **A**: Accuracy of both manual and hyper-optimized approximate contraction schemes for the 2D corner double line (CDL) model with lattice size of 64×64 and an effective $D = 2 \times 2 = 4$. **B**: schematic of the OBC CDL model.