

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

Johnnie Gray¹

Garnet Kin-Lic Chan¹

¹Division of Chemistry and Chemical Engineering, California Institute of Technology, Pasadena, USA 91125

The following material contains detailed information about the implementation and application of the hyper-optimized approximate contraction method. Sec. [A](#) details the generation of spanning trees. Sec. [B](#) derives the explicit form of the ‘compression’ projectors. In Sec. [C](#) we give details and pseudo-code for each of the tree building algorithms. Sec. [D](#) defines the estimated cost and size metrics and compares them. In Sec. [E](#) we define the various hand-coded approximate contraction schemes we compare to. Sec. [F](#) contains details about each model we apply approximate contraction to. In Sec. [G](#) we provide various details and extra results regarding the comparison to CATN [\[1\]](#). Finally in Sec. [H](#) we perform a brief study of the corner double line model.

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. [1](#). 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 1 r -local spanning tree

Input: graph G , initial region S_0 , max distance r

$\tau \leftarrow \{\}$ ▷ ordered set of pairs forming spanning tree
 $S \leftarrow S_0$ ▷ set of nodes spanned by the tree
 $c \leftarrow \{\}$ ▷ candidates to add to tree

for $u \in S$ **do** ▷ each node in original region
 for $v \in \text{NEIGHBORS}(G, u) \setminus S$ **do** ▷ connected nodes not in region
 $r_{uv} \leftarrow 1$ ▷ distance to original region
 $c \leftarrow c \cup \{(u, v, r_{uv})\}$
 end for
end for

while $|c| > 0$ **do**
 $u, v, r_{uv} \leftarrow \text{BEST}(c)$ ▷ pop the best candidate edge
 $c \leftarrow c \setminus \{(u, v, r_{uv})\}$
 if $(v \notin S) \wedge (r_{uv} \leq r)$ **then** ▷ node is new and close enough
 $S \leftarrow S \cup \{v\}$ ▷ add v to region
 $\tau \leftarrow \tau \cup \{(u, v)\}$ ▷ add edge to tree
 for $w \in \text{NEIGHBORS}(G, v) \setminus S$ **do** ▷ add new neighboring candidates
 $r_{vw} \leftarrow r_{uv} + 1$
 $c \leftarrow c \cup \{(v, w, r_{vw})\}$
 end for
 end if
end while

Return: τ, S

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 [2] 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 [3]. 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.

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

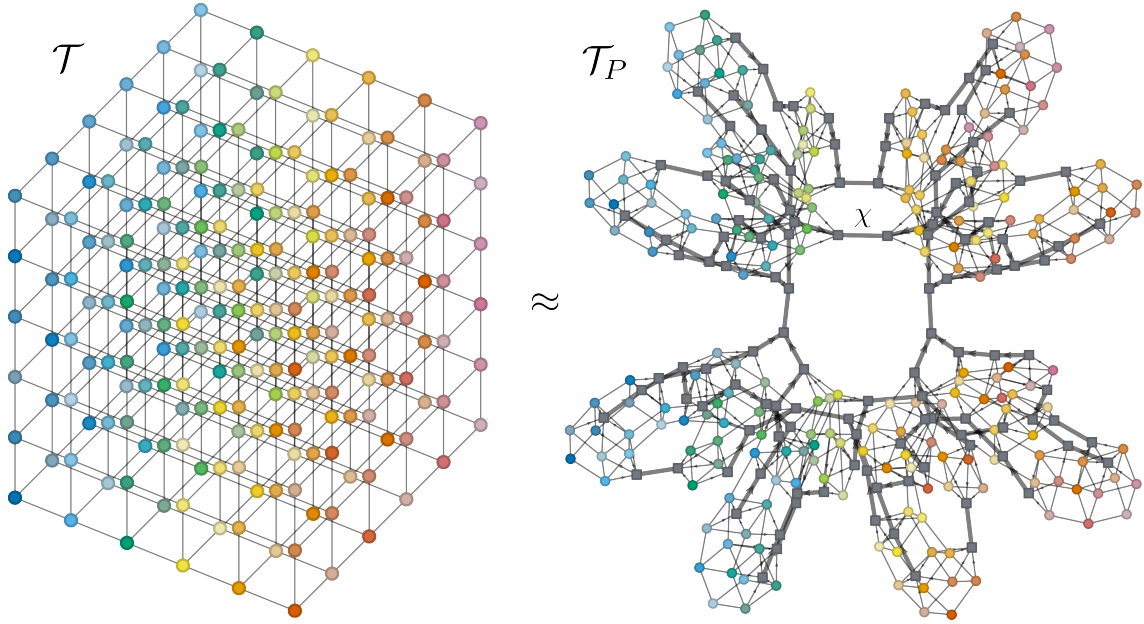


Figure 1: An example of transforming a tensor network, \mathcal{T} , into an exactly contractable tensor network, \mathcal{T}_P , using the explicit projector form of a approximate contraction tree. Here we take a $6 \times 6 \times 6$ with $D = 2$ cube and use an optimized contraction tree from the `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).

such as a local tree. Assuming we can decompose each into a orthogonal and ‘reduced’ 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:

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

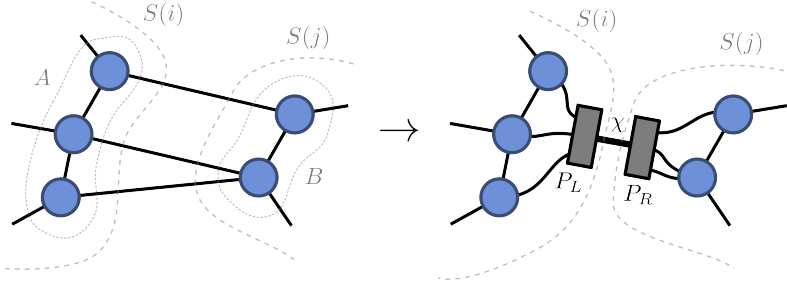
and likewise:

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

This form of the projectors makes explicit the equivalence to CTMRG and HOTRG [4, 5, 6], 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- r 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, \mathcal{T} , to another tensor network, \mathcal{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 \mathcal{T}_P then gives the approximate contracted value of \mathcal{T} . Such a mapping may be useful for relating the approximate contraction to other tensor network forms [7], or for performing some operations such as optimization [8]. 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. 1. 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 \mathcal{T}_P .

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 Υ . 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` [9] optimizer for this purpose. A scoring function computes some metric 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, y_{best} and Υ_{best} respectively. The result, outlined in Algorithm 2, 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 2 Hyper optimization loop

Input: graph G , max bond χ , builder B , optimizer O

$y_{best} \leftarrow \infty$

while *optimizing* **do**

$\bar{\theta} \leftarrow \text{SAMPLE_PARAMETERS}(O)$ ▷ Get new hyper parameters

$\Upsilon \leftarrow \text{GENERATE_TREE}(B, G, \bar{\theta})$ ▷ Build tree with new parameters

$y \leftarrow \text{SCORE_TREE}(\Upsilon, \chi)$ ▷ Score the tree

if $y < y_{best}$ **then**

$y_{best} \leftarrow y$

$\Upsilon_{best} \leftarrow \Upsilon$

end if

$\text{REPORT_PARAMETERS}(O, \bar{\theta}, y)$ ▷ Update optimizer with score

end while

Return: Υ_{best}

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 [10, 11], that assigns to each node a value according to how central it is in the network. This can be computed very efficiently as $c^{[v]} = \frac{1}{Z} \sum_{u \neq v} \frac{1}{\sqrt{d(u,v)+1}}$, where $d(u, v)$ is the shortest distance between nodes u and v . The normalization constant Z is chosen such that $c^{[v]} \in [0, 1] \forall v$.

C.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 θ_{new_size} .
- \log_2 size of new tensor *before* compression with weight θ_{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 χ_{greedy} to simulate the contraction with, which can thus deviate from the real value of χ 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.

C.2 Span

The `Span` algorithm builds an ordered contraction tree using a modified, tunable version of the spanning tree generator in Algorithm 1 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. 1), 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.

C.3 Agglom

The `Agglom` algorithm builds the contraction tree by repeated graph partitioning using the library `KaHyPar` [12, 13]. 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 ‘kml’,
- 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 χ_{agglom} (itself a hyper-parameter which can deviate from the real χ 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.

Algorithm 3 Branch and bound tree search

Input: graph G , Maximum bond dimension χ

$y_{best} \leftarrow \infty$

$c = \{\}$ ▷ candidate contractions

for $i, j \in \text{EDGES}(G)$ **do** ▷ populate with every pair of tensors

$y \leftarrow 0$ ▷ initial score

$p \leftarrow []$ ▷ the contraction ‘path’

$c \leftarrow c \cup \{(i, j, G, y, p)\}$

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** ▷ no need to explore further

continue

end if

if $|G| = 1$ **and** $y < y_{best}$ **then** ▷ finished contraction with best score

$y_{best} \leftarrow y$

$p_{best} \leftarrow p$

continue

end if

$p \leftarrow \text{APPEND}(p, (i, j))$ ▷ continue exploring

$(k, G, y) \leftarrow \text{SIMULATE_CONTRACTION}(i, j, G, \chi)$ ▷ k is the new node

for $l \in \text{NEIGHBORS}(G, k)$ **do** ▷ add new possible contractions

$c \leftarrow c \cup \{(k, l, G, y, p)\}$

end for

end while

$\Upsilon_{best} \leftarrow \text{BUILD_TREE_FROM_PATH}(G, p_{best})$

Return: Υ_{best}

C.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.

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 χ (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

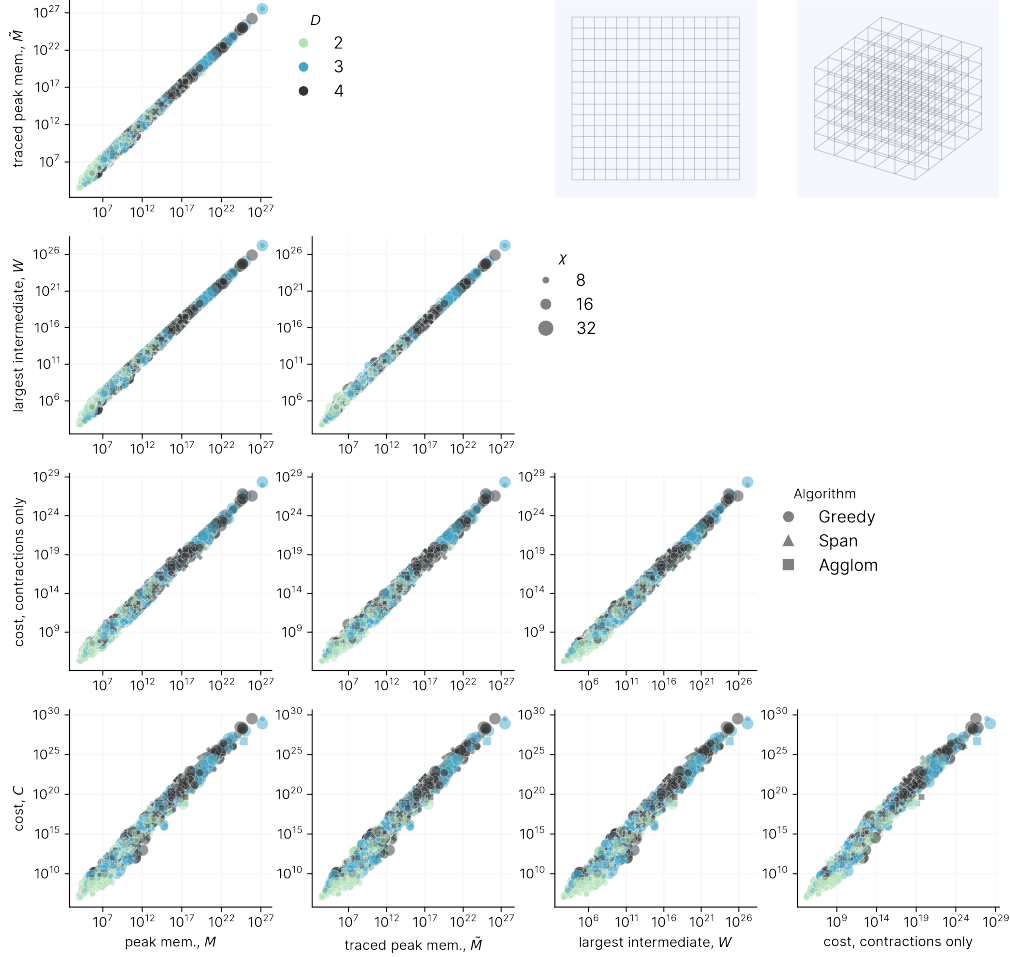


Figure 2: Relationship between various tree cost functions for randomly sampled approximate contraction trees for two geometries: 2D square of size 16×16 and 3D cube of size $6 \times 6 \times 6$ (pictured in insets). D , χ , the algorithm and its hyper-parameters are all uniformly sampled.

given by:

$$M = \max_{\alpha} \sum_{v \in \{v_{\alpha}\}} \text{size}(T^{[v]}) . \quad (1)$$

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 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 $m \geq n$: $4mn^2 - \frac{4}{3}n^3$.

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.

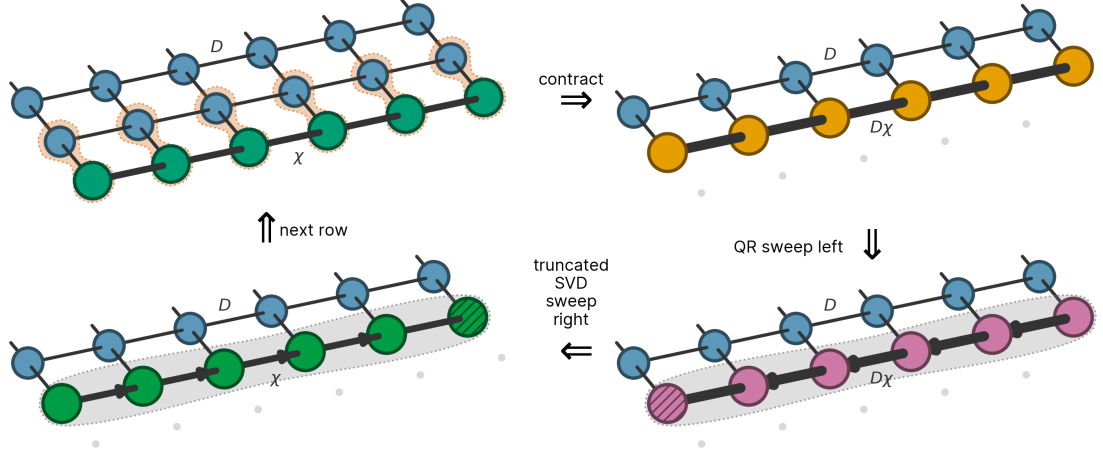


Figure 3: Overview of a single step of the manual 2D boundary contraction method that uses an MPS to sweep across the square.

In Fig. 2 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.

E Hand-coded Contraction Schemes

In Figs. 3-7 we illustrate the various hand-coded contraction schemes used as comparisons in the text: 2D boundary contraction, 2D corner transfer matrix RG [14], 2D higher-order TRG [15], 3D PEPS boundary contraction, and 3D higher-order TRG [15]. 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 [6] 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 [7] and the original references.

F Models

F.1 Ising Model

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

$$f = \frac{-\log Z}{N\beta} \quad (2)$$

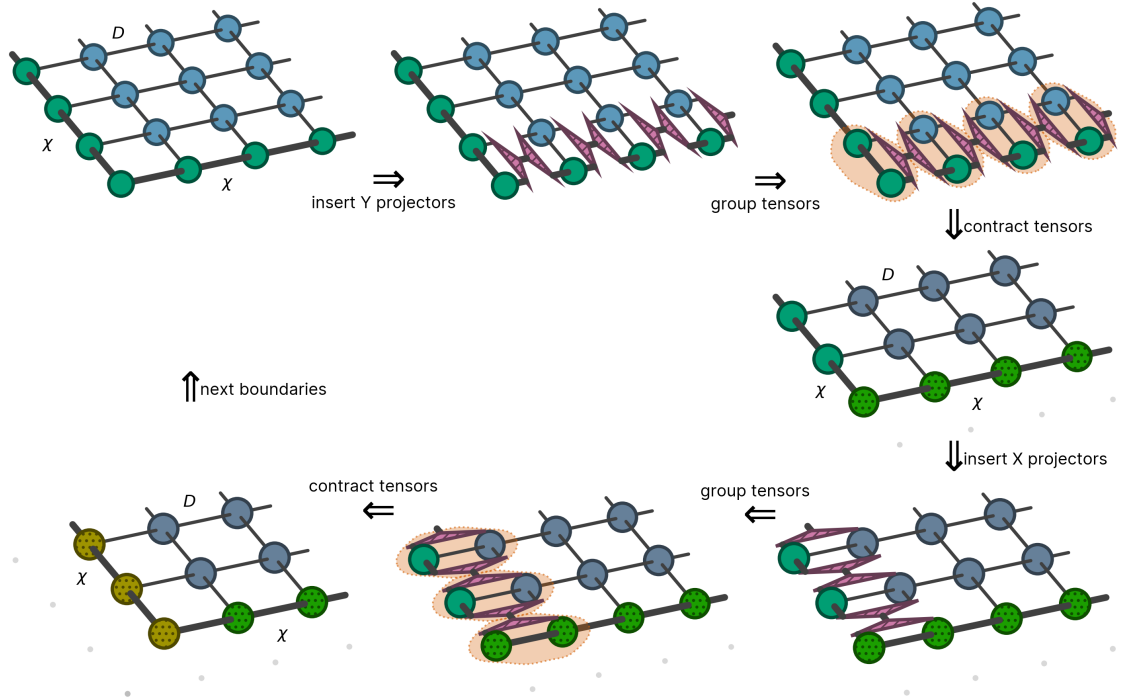


Figure 4: Illustration of two boundary contraction steps of CTMRG for a finite 2D lattice. The full algorithm proceeds to contract all four of the sides inwards in succession. Note that the projectors (pink) are not identical across the lattice but are computed specific to the local tensors to allow for finite in-homogeneous systems.

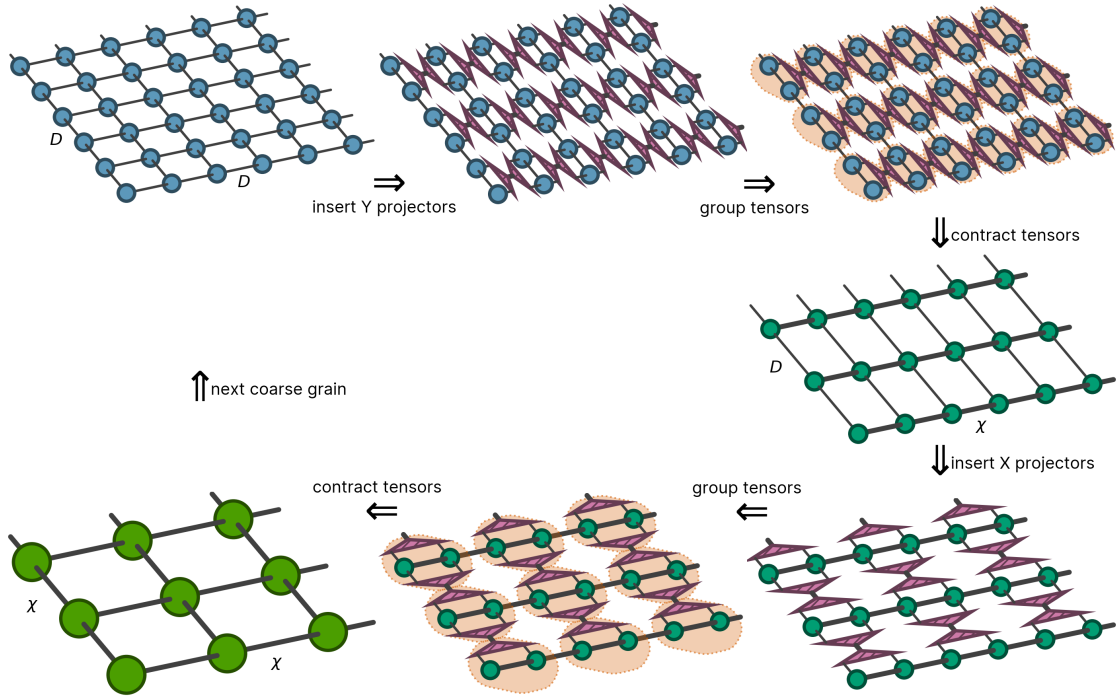


Figure 5: Illustration of a full coarse graining step of HOTRG for a finite 2D lattice. Note that once a round of coarse graining has taken place, all bonds are of size χ and so the next round starts with $D = \chi$. Note also that the projectors (pink) are not identical across the lattice but are computed specific to the local tensors to allow for finite in-homogeneous systems.

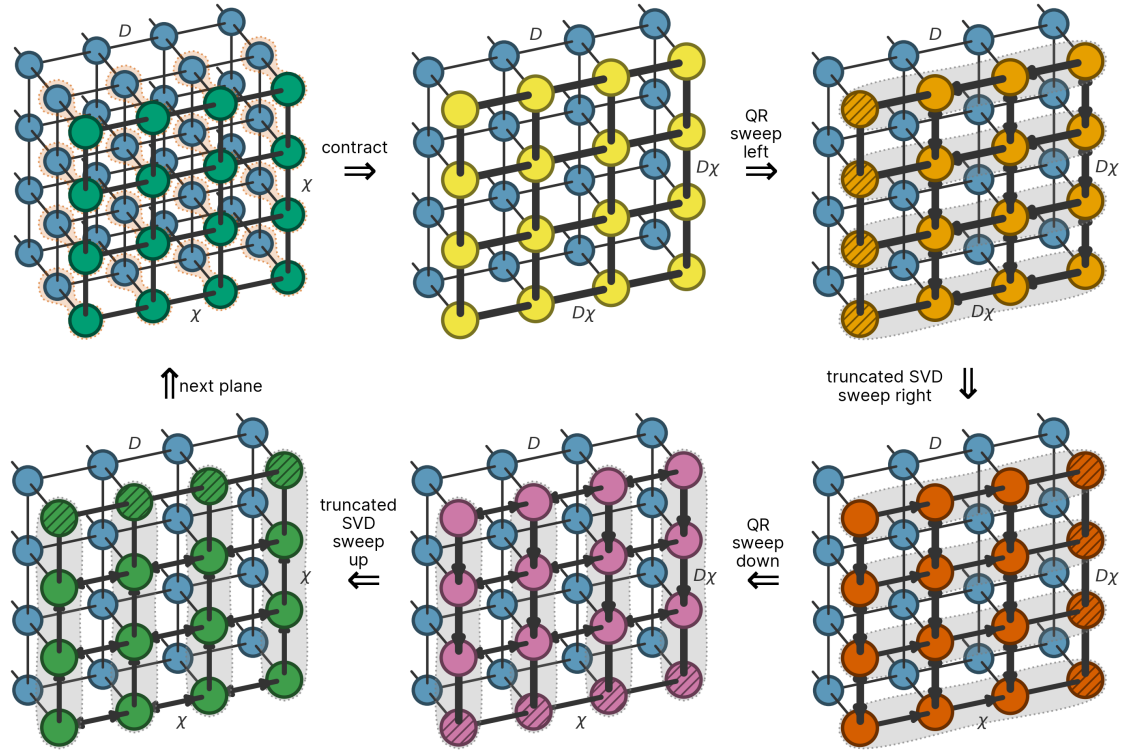


Figure 6: Illustration of a single step of the the manual 3D boundary contraction method that uses a PEPS to sweep across the cube. When L_x , L_y or $L_z = 1$ the scheme becomes equivalent to MPS boundary contraction.

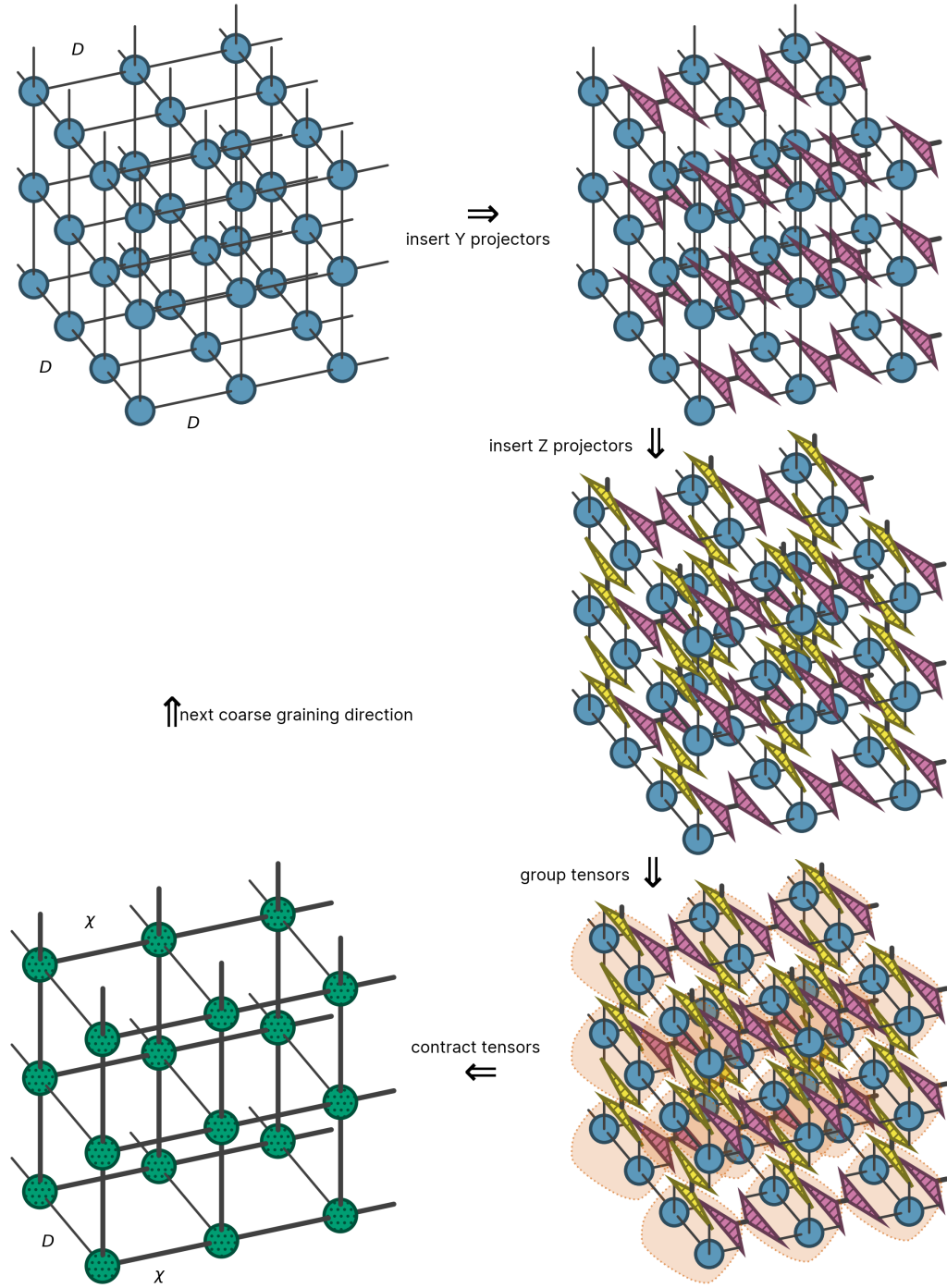


Figure 7: Illustration of a single coarse graining step of HOTRG for a finite 3D lattice. For brevity we only show coarse graining in the x-direction but the full algorithm coarse grains each of the three dimensions in succession. Note that once two or more directions have been coarse grained, all bonds will be of size χ and so subsequent rounds start with $D = \chi$. 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.

where the partition function, Z , is given by:

$$Z = \sum_{\{\sigma\}} \prod_{\langle i,j \rangle} \exp(j\beta\sigma_i\sigma_j), \quad (3)$$

$\sigma_k \in [1, -1]$ 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 many quantities in such models, we note that the partition function and free energy are typically much more challenging [16]. 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,

$$T_{\{e_v\}}^{[v]} = \sum_i \prod_{e_j \in \{e_v\}} W_{i,e_j} \quad (4)$$

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

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

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. (3) 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:

$$\Delta f = \left| 1 - \frac{f}{f_{\text{exact}}} \right| = \left| 1 - \frac{\log Z}{\log Z_{\text{exact}}} \right| \quad (5)$$

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, $\beta_c = \frac{\log(1+\sqrt{2})}{2} \approx 0.44$ [17, 18].

F.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.

F.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), which can be formed as a tensor network with the following tensor on each vertex:

$$T_{i,j,k,\dots} = \begin{cases} 1, & \text{if } i + j + k + \dots = 1 \\ 0, & \text{otherwise} \end{cases} . \quad (6)$$

The total number of valid configurations is then the contraction:

$$W = \sum_{\{e\}} \prod_v T_{\{e_v\}}^{[v]} \quad (7)$$

This is also equivalent the counting problem positive #1-IN-3SAT [19, 20, 21], the decision version of which is NP-Complete [22, 23]. 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) [19, 20]. The residual entropy per site is given by:

$$S = \frac{\log W}{n} , \quad (8)$$

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

$$\Delta S = 1 - \frac{S}{S_{\text{exact}}} . \quad (9)$$

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 [24]. 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

$$\bar{W} = (\sqrt{2} + O(n^{-\frac{2}{3}})) e^{\frac{1}{4}} ((k-1)^{k-1} / k^{k-2})^{n/2} . \quad (10)$$

If we take the limit of this we find:

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

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

G Performance comparison to CATN

So far where appropriate we have compared our method to manually specified contraction orders in 2D and 3D. In [1], 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

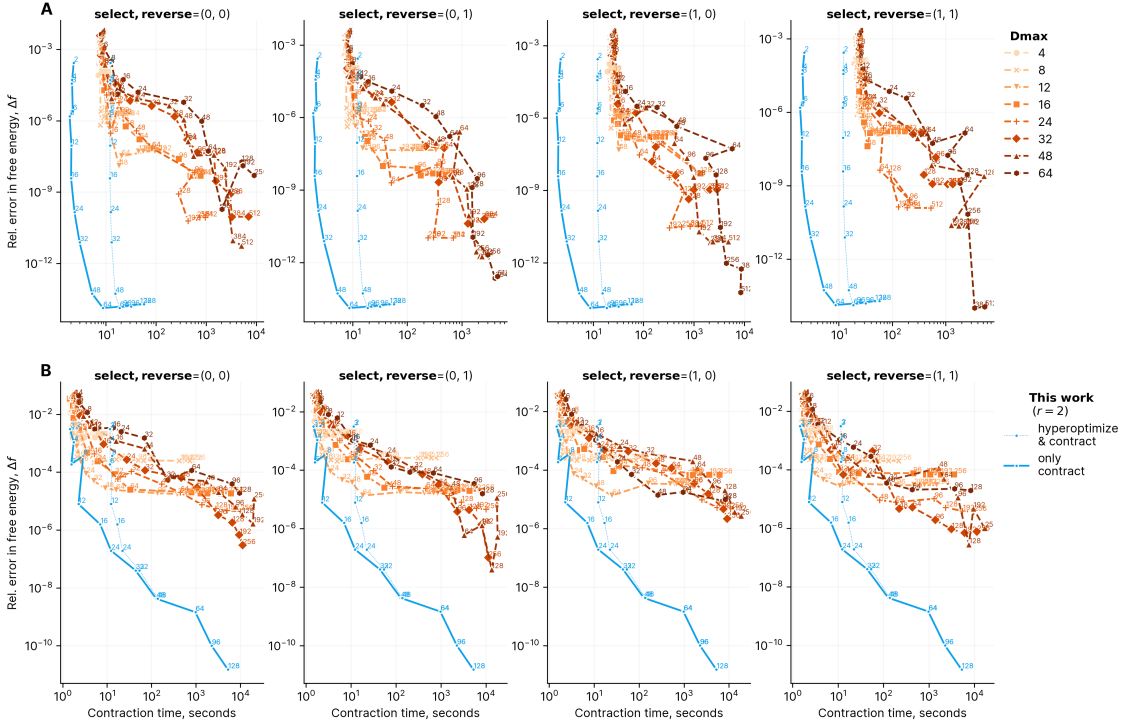


Figure 8: Performance comparison of this current work (blue) and the algorithm of Pan et al. [1] (orange) using various settings for computing the free energy of the Ising model at approximately the critical point of **A**: a 32×32 square lattice at the approximate critical temperature $\beta=0.44$ and **B**: a $6 \times 6 \times 6$ cubic lattice at the approximate critical temperature $\beta=0.3$. For both algorithms χ 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).

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 [1] is available at [25] 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. 8 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 χ 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 χ , and two main parameters controlling how to select the bonds to remove, *select* and *reverse*. The relationship between D_{\max} and χ 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

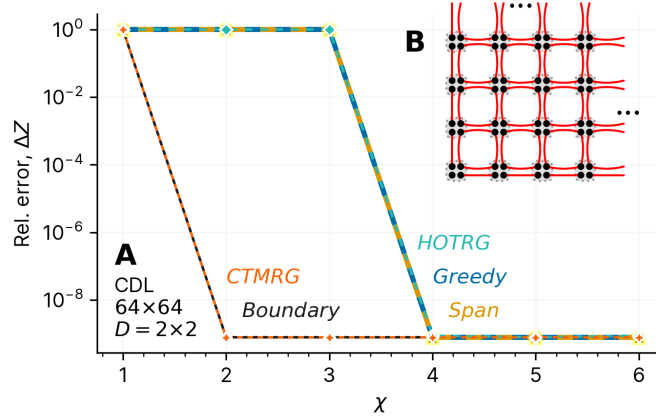


Figure 9: **A**: Accuracy of both manual and hyper-optimized approximate contraction schemes for the 2D corner double line (CDL) model with lattice size of 64×64 and an effective $D = 2 \times 2 = 4$. **B**: schematic of the OBC CDL model.

`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` [26], and we take the best time out of three repeats.

In both the 2D and 3D cases, Figs. 8A 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 [1] 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.

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 [27]. 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 [28] and also more advanced algorithms such as HOTRG [15] 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, and the loop bond dimension is not too large 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 χ is above some very small threshold. In Fig. 9A 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. 9B 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 . Nonetheless, for the purposes of studying the entanglement renormalization flow (as well as to handle loop correlations with larger bond dimension) it will be interesting to generalize our contraction schemes to include disentanglers. This will be studied in future work.

References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] L. Wang and F. Verstraete, “Cluster update for tensor network states,” Oct. 2011.
- [5] 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.
- [6] S. Iino, S. Morita, and N. Kawashima, “Boundary Tensor Renormalization Group,” *Physical Review B*, vol. 100, p. 035449, July 2019.
- [7] S.-J. Ran, E. Tirrito, C. Peng, X. Chen, L. Tagliacozzo, G. Su, and M. Lewenstein, *Tensor network contractions: methods and applications to quantum many-body systems*. Springer Nature, 2020.
- [8] 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.
- [9] J. Rapin and O. Teytaud, “Nevergrad - A gradient-free optimization platform.” <https://GitHub.com/FacebookResearch/Nevergrad>, 2018.
- [10] M. A. Beauchamp, “An improved index of centrality,” *Behavioral science*, vol. 10, no. 2, pp. 161–163, 1965.
- [11] 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.
- [12] S. Schlag, *High-Quality Hypergraph Partitioning*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020.
- [13] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, “High-quality hypergraph partitioning,” *ACM J. Exp. Algorithmics*, mar 2022.
- [14] 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.

- [15] 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.
- [16] 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.
- [17] L. Onsager, “Crystal statistics. i. a two-dimensional model with an order-disorder transition,” *Physical Review*, vol. 65, no. 3-4, p. 117, 1944.
- [18] R. J. Baxter, *Exactly solved models in statistical mechanics*. Elsevier, 2016.
- [19] 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.
- [20] 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.
- [21] S. Kourtis, C. Chamon, E. Mucciolo, and A. Ruckenstein, “Fast counting with tensor networks,” *SciPost Physics*, vol. 7, no. 5, p. 060, 2019.
- [22] 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.
- [23] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1979.
- [24] 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.
- [25] F. Pan, P. Zhou, S. Li, and P. Zhang, “CATN.” <https://github.com/panzhang83/catn>, 2019.
- [26] 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.
- [27] G. Evenbly and G. Vidal, “Tensor Network Renormalization,” *Physical Review Letters*, vol. 115, p. 180405, Oct. 2015.
- [28] 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.