The Tensor Renormalization Group

Caleb Cook
Department of Physics, Harvard University, Cambridge MA 02138
(Dated: May 15, 2015)

We review the tensor renormalization group (TRG), a real space renormalization technique for coarse-graining 2D classical lattice models. We outline the method briefly in the context of a honeycomb lattice and then show explicitly how tensor renormalization proceeds on a square lattice. An implementation of the TRG on a square lattice Ising model is given in Python, and a comparison between the exact and TRG-obtained free energies of the model is made. Finally, we discuss the limitations of the TRG and an extension of the method that has been introduced to address these limitations.

I. INTRODUCTION

Understanding emergent phenomena in many-body systems is one of the major goals of modern physics, and even numerically simulating such systems on computers remains a great challenge. If the microscopic degrees of freedom and their interactions are known, one may write the partition function of a classical statistical system (a weighted sum over all microscopic configurations), or the analogous path integral of a quantum many-body system (a weighted sum over all conceivable trajectories). These objects describe the collective physical properties of a many-body system completely, but are in general very difficult to express in closed form. In spite of this difficulty, the spin-blocking prescription proposed by Kadanoff [1] and subsequent implementation by Wilson [2] have opened the path to non-perturbative approaches based on coarse-graining a lattice. More recently, Levin and Nave have proposed the *tensor renormalization group* [3] as a more general RG approach for classical lattice models.

The tensor renormalization group may been seen as a generalization of the *density matrix renormalization group* (DMRG) method, introduced by White [4] to study the ground state of Heisenberg spin chains. The DMRG itself generalizes Wilsonian RG by reformulating the approach in terms of matrix product states, iteratively dividing the system into blocks and truncating basis states at each step. This truncation makes the algorithm tractable and is performed in an error-minimizing way by maximizing the entanglement entropy

$$S = -\sum_l \lambda_l \ln \lambda_l \tag{1}$$

at each step, where λ_l are the the eigenvalues of the reduced (post-truncation) density matrix. The DMRG has been widely used to study thermodynamic properties of quantum lattice models, due in large part to its lack of "minus sign" problem. This "minus sign" problem plagues the otherwise most powerful numerical method, quantum Monte Carlo, and makes it inapplicable to most fermion systems [5].

Although approximate due to these truncations, the DMRG is an extraordinarily precise method in one-dimensional systems. This is due to the fact that in one dimension, the surface of a lattice model contains just two points and does not grow with system size. As the entanglement entropy scales with this surface area, only a small matrix dimension is necessary for a matrix product state to accurately represent a quantum state. In higher dimensions, exponential growth of the matrix dimension at each iteration is required to faithfully represent a quantum state, making the algorithm intractable. Addressing this breakdown of the DRMG, Levin and Nave [3] introduced the tensor renormalization group as way of generalizing the DMRG to higher dimensions, in the (simpler) context of classical systems.

In Section II, we briefly outline the TRG method using a honeycomb lattice as an example, following [3]. In Sections III and V, we discuss in more detail the particular application of the TRG to a square lattice and its implementation in Python. In Section IV, we compare the computed free energy of this TRG implementation to the exact result and discuss how the TRG can be improved.

II. TRG OVERVIEW

The tensor renormalization group (TRG) is a real space renormalization technique for coarse-graining 2D classical lattice models. In the TRG, one begins by writing the given lattice's associated *tensor network*: a network with tensors $T_{ijk\ldots}$ and D-level indices $i = 1, 2, \ldots, D$ at each node for which

$$Z = \sum_{i_\alpha, j_\alpha, k_\alpha \ldots} T_{i_1 j_1 k_1 \ldots} T_{i_2 j_2 k_2 \ldots} \cdots = \mathrm{tTr}\left[\otimes_{i=1}^{N} T\right] \tag{2}$$

where $\otimes_i$ is a tensor product over all network nodes and the tensor-trace tTr represents the sum over all tensor indices. A given lattice may have multiple associated tensor networks. Moreover, by making a duality transformation and considering Boltzmann weights in dual lattice variables, one can represent any classical lattice model with local interactions as a tensor network. In dimensions two and higher, carrying out the defining tensor-trace has been shown to be NP-hard [6]. The TRG is an approximate method that proceeds by making a DMRG-like truncation at each step, allowing for tractable compu-

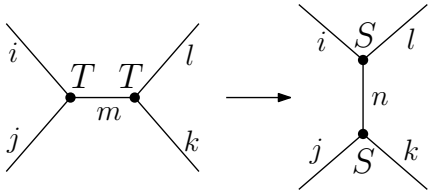

FIG. 1. Graphical representation of the rewiring procedure in the TRG of a rank-3 tensor.

tation of the partition function.

On a honeycomb lattice, each site has three nearest neighbors, and so we use a rank-3 tensor T_{ijk} in the network representation. One iteration of the TRG then consists two steps: rewiring and decimation. In the first step, we find a tensor S_{lin} satisfying

$$\sum_m T_{ijm}T_{klm} \approx \sum_n S_{lin}S_{jkn} \tag{3}$$

This corresponds to a rewiring of the bonds in the lattice, as depicted in Figure 1. This step is (in general) approximate. The optimal tensor S is found by viewing the left-hand side as a $D^2 \times D^2$ matrix $M_{li,jk}$, performing singular value decomposition (SVD), and then discarding all but the $D' = \min\left(D^2, D_{\text{cut}}\right)$ largest singular values to obtain the $D'^2 \times D'$ matrix $S_{li,n}$. Here, D_{cut} is a fixed parameter of the algorithm that specifies the truncation threshold of D over all iterations, thus keeping the dimensionality of T from increasing without bound.

The second step then involves decimating the extraneous triangles generated by the rewiring procedure, as shown in Figure 2, by summing over their internal indices. This step is exact, and the contraction of triangle indices yields a coarser honeycomb lattice's tensor network with rank-3, D'-dimensional tensors

$$T'_{ijk} = \sum_{pqr} S_{kpq}S_{jqr}S_{irp} \tag{4}$$

at each node. It is in this way that the mapping $T \to T'$ defines a coarse-graining RG flow on the tensor network for which

$$Z = \text{tTr}\left[\otimes_{i=1}^{N} T\right] \approx \text{tTr}\left[\otimes_{i'=1}^{N/3} T'\right] \tag{5}$$

Repeated iteration of the TRG eventually yields a single unit cell possessing six sites/tensors, in which case the tensor-trace above may be easily evaluated.

Error in the TRG is introduced via the approximate SVD decomposition in the rewiring step. Utilizing the mapping between 2D classical models and (1+1)D quantum models and the entanglement properties of gapped ground states, Levin and Nave [3] show that this error scales as

$$\left|M - S \cdot S^T\right|^2 \sim \exp\left[-\text{const} \cdot (\log D_{\text{cut}})^2\right] \tag{6}$$

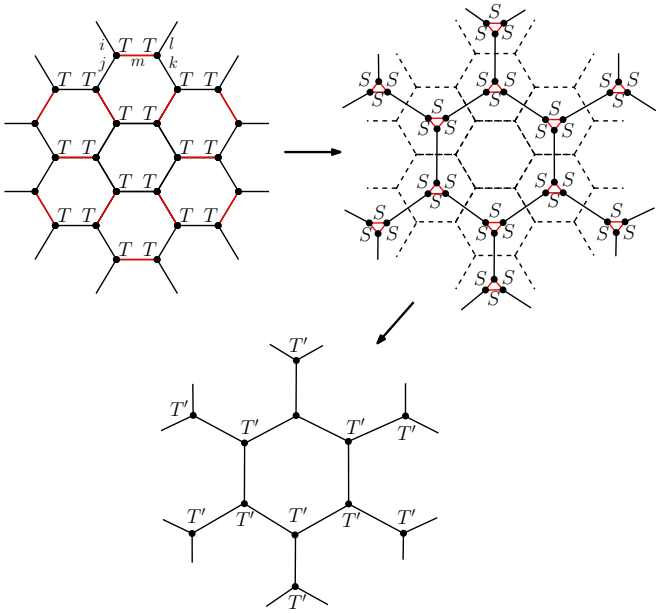

FIG. 2. One iteration of the TRG on a honeycomb lattice's tensor network. Red bonds in network are rewired per the decomposition shown in Figure 1, and red triangles in the rewired network are subsequently decimated. After rewiring and decimation, the number of nodes has been reduced by a factor of 3, and a coarser honeycomb lattice's tensor network is obtained.

away from criticality. By choosing D_{cut} sufficiently large, we can make this error arbitrarily small. It is this property ensures the accuracy of the TRG method.

III. 2D SQUARE LATTICE ISING MODEL

In [3], Levin and Nave mention in passing that a TRG approach is also possible on a square lattice, but do not show explicitly how to implement such a method. We therefore turn our attention to applying the TRG to a 2D square lattice of Ising spins $s_i = \pm 1$ with Hamiltonian

$$H = -J \sum_{\langle ij \rangle} s_i s_j \tag{7}$$

The Python code underlying this implementation of the TRG is shown in the Appendix: Section V.

A. Tensor Network Representation

The partition function of this model is given by

$$Z = \text{Tr}\, e^{-\beta H} = \text{Tr} \prod_{\square_{ijkl}} e^{\beta J(s_i s_j + s_j s_k + s_k s_l + s_l s_i)/2} \tag{8}$$

where the trace here sums over all spin configurations $\{s_i\}$ of the lattice, and the product is taken over all

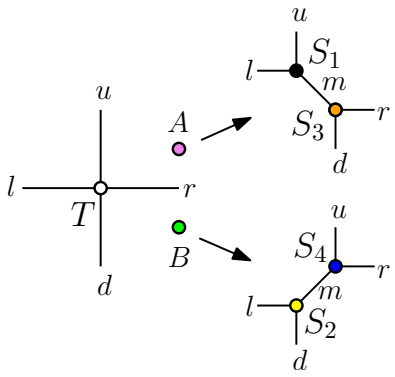

FIG. 3. Graphical representation of the rewiring procedure in the TRG of a rank-4 tensor, keeping in mind the sublattice from which each tensor came.

square plaquettes in the lattice. The factor of 1/2 in the above exponent comes from the fact that breaking the lattice into squares double counts each bond. Switching to bond variables $\sigma_{ij} = s_i s_j = \pm 1$ then gives

$$Z = \mathrm{Tr} \prod_{\langle ij \rangle} \delta(\sigma_{ij} - s_i s_j) \prod_{\square_{ijkl}} e^{\beta J(\sigma_{ij}+\sigma_{jk}+\sigma_{kl}+\sigma_{li})/2} \quad (9)$$

where the trace here now sums over all spins s_i and bonds σ_{ij}. The delta function in the above expression gives zero weight to unphysical configurations in which the product of bonds around a loop has value -1. We can therefore rewrite this delta function to obtain

$$Z = \mathrm{Tr} \prod_{\square_{ijkl}} \frac{1 + \sigma_{ij}\sigma_{jk}\sigma_{kl}\sigma_{li}}{2} e^{\beta J(\sigma_{ij}+\sigma_{jk}+\sigma_{kl}+\sigma_{li})/2} \quad (10)$$

Now consider the dual lattice, which is itself a square lattice formed by connecting perpendicular bond bisectors in the original lattice. There is a one-to-one correspondence between square plaquettes $\square_{ijkl}$ in the original lattice and sites p in the dual lattice. As such, we can take our tensor network to be the square lattice's dual lattice and write

$$Z = \mathrm{tTr} \left[\otimes_{p=1}^{N} T \right] \quad (11)$$

with tensors

$$T_{r_p u_p l_p d_p} = \frac{1 + \sigma_r^p \sigma_u^p \sigma_l^p \sigma_d^p}{2} e^{\beta J(\sigma_r^p + \sigma_u^p + \sigma_l^p + \sigma_d^p)/2} \quad (12)$$

associated with each network node p. In this definition, $r_p, u_p, l_p, d_p = 0, 1$ index the rightward, upward, leftward, and downward bonds, respectively, exiting each node p. A bond index $x_p = 0$ corresponds to anti-aligned spin variables $\sigma_{x_p} = -1$, and a bond index $x_p = 1$ corresponds to aligned spin variables $\sigma_{x_p} = 1$.

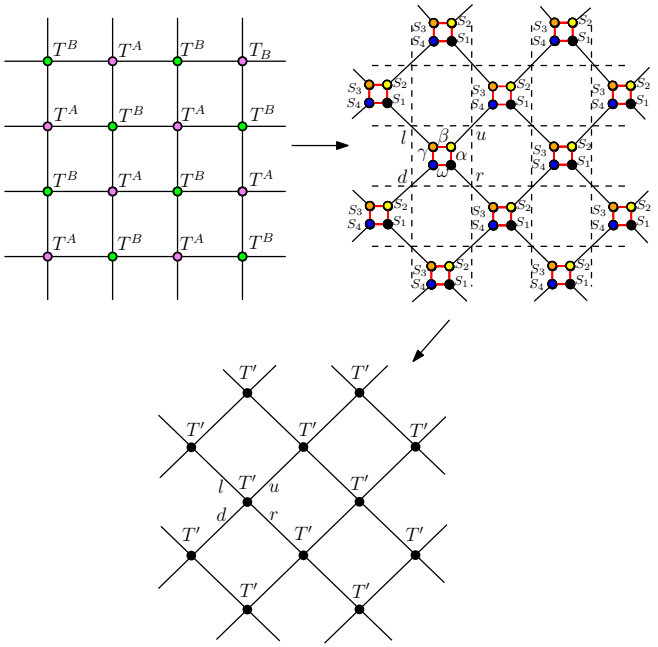

FIG. 4. One iteration of the TRG on a square lattice's tensor network. Each sublattice in the network is rewired per the decomposition shown in Figure 3, and red squares in the rewired network are subsequently decimated. After rewiring and decimation, the number of nodes has been reduced by a factor of 4, and a coarser square lattice's tensor network is obtained.

B. Rewiring and Decimation

We now rewire the tensor network just obtained by decomposing each rank-4 tensor T into the two rank-3 tensors S_i and S_{i+2}, as shown in Figure 3. In [3], Levin and Nave note a phase ambiguity that appears when associating rewired tensors S with SVD components of T. This ambiguity can be avoided if one keeps track of different tensors T^A and T^B for each sublattice of the tensor network. Consequently, we decompose the tensor T separately on each sublattice as

$$T^A_{ruld} = \sum_{m=1}^{D^2} S_{1,ulm} S_{3,drm} \quad (13)$$

$$T^B_{ruld} = \sum_{m=1}^{D^2} S_{2,ldm} S_{4,rum} \quad (14)$$

The tensors S_i are obtained approximately by SVD. Explicitly, we view $T^A_{ruld} = M_{lu,rd}$ as a $D^2 \times D^2$ matrix and perform SVD to obtain

$$M = U \Sigma V^\dagger \quad (15)$$

with Σ diagonal, having D^2 eigenvalues $\lambda_1 > \lambda_2 > \cdots$.

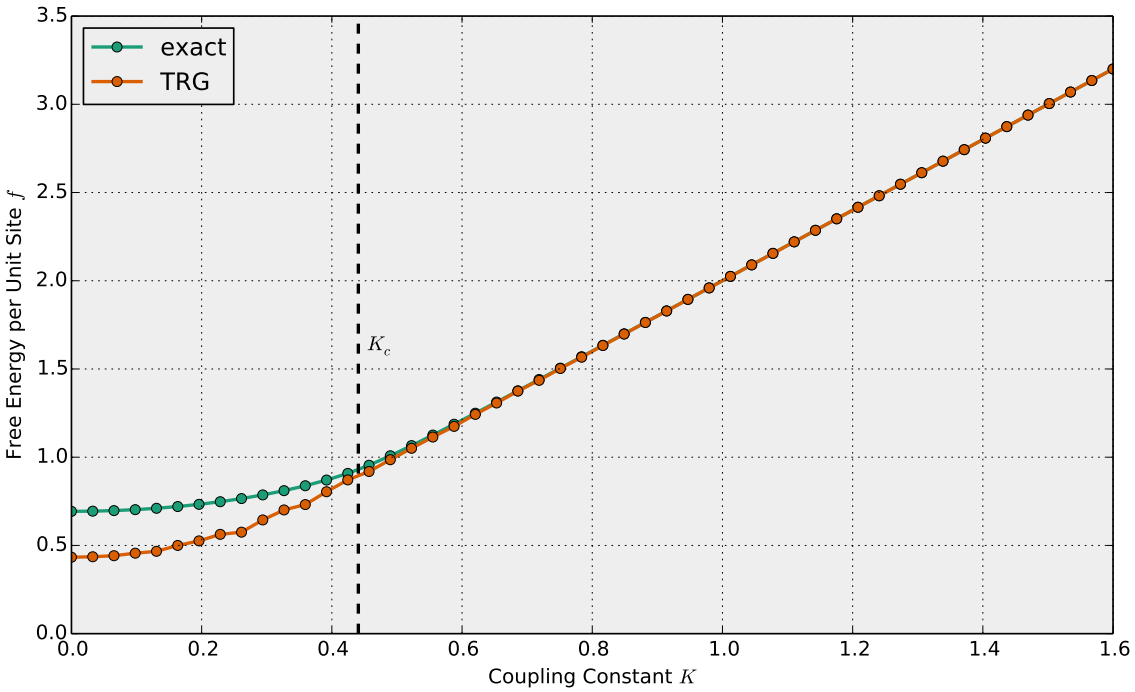

FIG. 5. Comparison of the exact and TRG-obtained free energies of a 2D square lattice Ising model. The TRG was applied 3 times with $D_{\text{cut}} = 6$. The accuracy of the TRG method is seen to decrease by several orders of magnitude as the critical point K_c is approached.

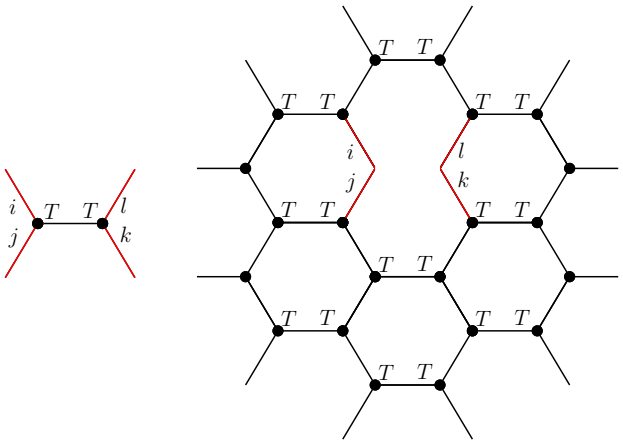

FIG. 6. Example of the "environment" neglected by the TRG but taken into account by the SRG on a honeycomb lattice. A pair of tensors (left) to be decomposed in the rewiring step of the TRG and their environment (right) are shown.

We may then write

$$T^A_{ruld} \approx \sum_{m=1}^{D'} \underbrace{\left(\sqrt{\lambda_m} U_{lu,m}\right)}_{S_{1,ulm}} \underbrace{\left(\sqrt{\lambda_m} V^\dagger_{m,rd}\right)}_{S_{3,drm}} \qquad (16)$$

which is approximate since we have only kept a fixed number $D' = \min\left(D^2, D_{\text{cut}}\right)$ of the largest eigenvalues λ_m. The decomposition of T^B is performed nearly identically.

Decimation is then carried out by summing over the internal indices of each extraneous square generated by the network rewiring, shown in Figure 4. This generates a coarser square lattice's tensor network with rank-4, D'-dimensional tensors

$$T'_{ruld} = \sum_{\alpha,\beta,\gamma,\omega=1}^{D} S_{1,\omega\alpha r} S_{2,\alpha\beta u} S_{3,\beta\gamma l} S_{4,\gamma\omega d} \qquad (17)$$

at each node. The iteration $T \to T'$ then defines the TRG flow on this model.

IV. DISCUSSION

The TRG algorithm described in the previous section was implemented in Python, shown in Appendix: Section V. As Python is a high-level programming language, the code is fairly human readable but runs rather slowly. We were therefore only able to run 3 iterations of the TRG on the square lattice with $D_{\text{cut}} = 6$. Even with $D_{\text{cut}} = 6$, good agreement between the exact and TRG free energies is obtained, as can be seen in Figure 5.

The TRG error scaling in Eq. 6 was derived by Levin and Nave [3] under the assumption that the system is not critical. Near criticality, the associated quantum states under the classical-quantum lattice model mapping become gapless ground states, which are more entangled then their gapped counterparts. This results in the truncated SVD decomposition failing to accurately represent the original tensor T near criticality. The 2D square lattice Ising model experiences a phase transition and therefore becomes critical at the value [7],

$$K_c = \frac{1}{2}\ln\left(1+\sqrt{2}\right) \approx 0.441 \tag{18}$$

The breakdown of the TRG near this phase transition can be seen in Figure 5.

Even away from criticality, the TRG can be improved. As noted by Xiang and others [8], the TRG fails to take into account the "environment" lattice when performing SVD composition tensors node in the rewiring step. More specifically, at each step the TRG minimizes the truncation error of the local matrices M, whereas it is in fact the truncation error of entire partition function, which includes a contribution M_e from the environment,

$$Z = \mathrm{tTr}\left[MM_e\right] \tag{19}$$

that should be minimized. The concept of an "environment" is shown graphically in Figure 6. In [8], this extension of the TRG is called the second renormalization group (SRG) and is shown to improve the accuracy of the TRG by several orders of magnitude.

ACKNOWLEDGMENTS

I would like to thank Prof. Mehran Kardar for teaching his challenging but rewarding MIT 8.334 course. I would also like to thank Prof. Guifre Vidal for his helpful guidance on this project via email correspondence.

[1] L. P. Kadanoff. Scaling laws for ising models near tc. *Physics*, 2(6):263–272, 1966.
[2] Kenneth K. G. Wilson. The renormalization group: Critical phenomena and the kondo problem. *Rev. Mod. Phys.*, 47:773–840, Oct 1975.
[3] M. Levin and C. P. Nave. Tensor renormalization group approach to two-dimensional classical lattice models. *Phys. Rev. Lett.*, 99:120601, Sep 2007.
[4] S. R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69:2863–2866, Nov 1992.
[5] M. Troyer and U. Wiese. Computational complexity and fundamental limitations to fermionic quantum monte carlo simulations. *Phys. Rev. Lett.*, 94:170201, May 2005.
[6] N. Schuch. Computational complexity of projected entangled pair states. *Physical Review Letters*, 98(14), 2007.
[7] M. Kardar. *Statistical physics of fields*. Cambridge University Press, 2007.
[8] H. H. Zhao, Z. Y. Xie, Q. N. Chen, Z. C. Wei, J. W. Cai, and T. Xiang. Renormalization of tensor-network states. *Phys. Rev. B*, 81:174411, May 2010.

V. APPENDIX: PYTHON IMPLEMENTATION

```python
# TRG.py

# Tensor Renormalization Group applied to a 2D square lattice Ising model

import numpy as np
from numpy.linalg import svd
from itertools import product

# returns TRG-obtained partition function Z
def Z_TRG(K, Dcut, no_iter):
    D = 2
    inds = np.arange(D)

    # set up initial tensor network
    T = np.empty([D, D, D, D])
    for r, u, l, d in product(inds, inds, inds, inds):
        T[r][u][l][d] = .5*(1 + (2*r-1)*(2*u-1)*(2*l-1)*(2*d-1))*np.exp(2*K*(r+u+l+d-2))

    for n in np.arange(no_iter):
        D_new = min(D**2, Dcut)
        inds_new = np.arange(D_new)

        # perform SVD decomposition and rewiring on each sublattice
        Ma, Mb = [np.empty([D**2, D**2])]*2
        for r, u, l, d in product(inds, inds, inds, inds):
            Ma[l + D*u][r + D*d] = T[r][u][l][d]
            Mb[l + D*d][r + D*u] = T[r][u][l][d]

        S1, S2, S3, S4 = [np.empty([D, D, D_new])]*4

        U, L, V = svd(Ma)
        L = np.sort(L)[::-1][0:D_new]
        for x, y, m in product(inds, inds, inds_new):
            S1[x, y, m] = np.sqrt(L[m])*U[x + D*y][m]
            S3[x, y, m] = np.sqrt(L[m])*V[m][x + D*y]

        U, L, V = svd(Mb)
        L = np.sort(L)[::-1][0:D_new]
        for x, y, m in product(inds, inds, inds_new):
            S2[x, y, m] = np.sqrt(L[m])*U[x + D*y][m]
            S4[x, y, m] = np.sqrt(L[m])*V[m][x + D*y]

        # decimate extraneous, TRG-generated squares to obtain T'
        T_new = np.empty([D_new, D_new, D_new, D_new])

        for r, u, l, d in product(inds_new, inds_new, inds_new, inds_new):
            T_new[r][u][l][d] = 0
            for a, b, g, w in product(inds, inds, inds, inds):
                T_new[r][u][l][d] += S1[w,a,r]*S2[a,b,u]*S3[b,g,l]*S4[g,w,d]

        D = D_new
        inds = inds_new
        T = T_new

    # after final TRG iteration, trace out remaining tensor to obtain Z
    Z = 0
    for r, u, l, d in product(inds, inds, inds, inds):
        Z += T[r][u][l][d]

    return Z
```