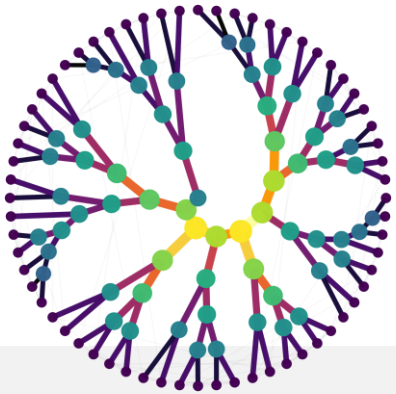


Hyper-optimized tensor network contraction - simplifications, applications and approximations

1st April 2021

Johnnie Gray

Garnet Chan Group

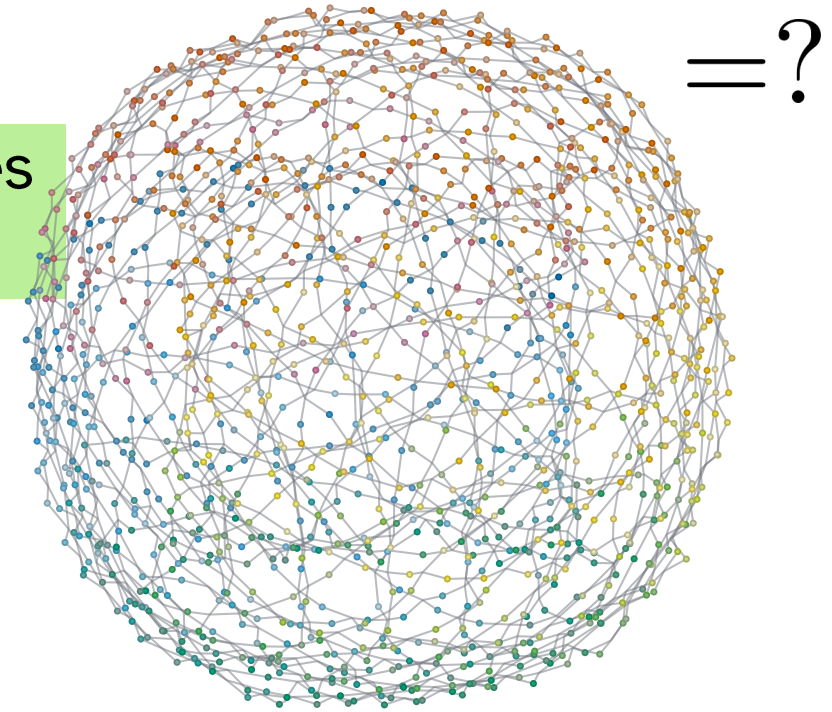
Caltech

mainly joint work with **Stefanos Kourtis**: arXiv:2002.01935

Rough setting + overview:

Some fundamental questions we can ask of tensor networks /decompositions:

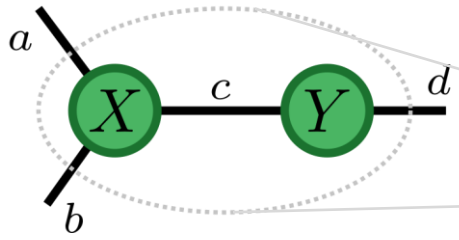
- can a tensor network **represent** X?
- can we **find** that representation?
- given a tensor network that exactly describes an unknown tensor, can we **compute** it?



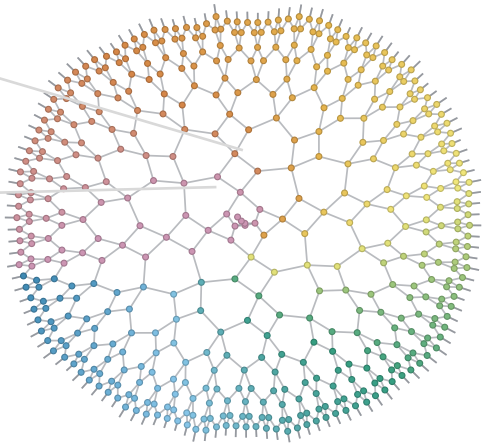
1. Frame task as nested hypergraph partitioning
2. Use this to motivate ‘simplification’ rules
3. Extend to approximate contraction?

Tensor networks – general defn

$T_{abd} = \sum_c X_{abc} Y_{cd}$



$G(V, E)$



Contraction of tensor network:

$$c(\mathcal{T}) = \sum_{x \in \mathcal{X}(E)} \prod_{u \in V} T_u[\overrightarrow{x_u}]$$

Sum over all, exponentially many index strings.

Quantum computation
- T_u ~ gates, states

Many-body quantum
- T_u ~ virtual Hilbert spaces

SAT / Counting
- T_u are clauses, constraints

Classical Stat Mech
- T_u describe local energies

Inference
- T_u are conditional prob dists

...

Exact tensor network contraction

Can describe huge tensors very efficiently… but actually contracting them generally hard (#P-Complete). Nonetheless can exponentially improve on naïve sum:

$$\sum_{x=abcdefgh} T_0(x_{abc})T_1(x_{adef})T_2(x_{bfg})T_3(x_{cdh})T_4(x_{egh})$$

By making use of associativity to 'bubble' tensors:

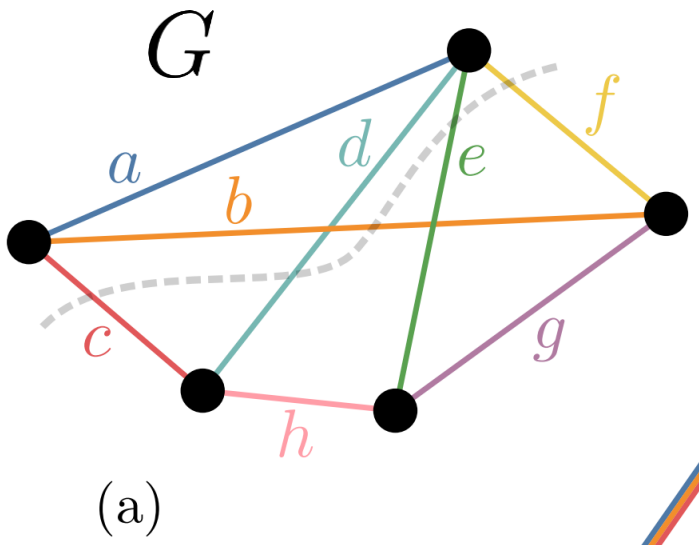
$$\left(\sum_{abc} T_0(x_{abc}) \left(\sum_{def} T_1(x_{adef}) \left(\sum_{g} T_2(x_{bfg}) \left(\sum_{h} T_3(x_{cdh})T_4(x_{egh})\right)\right)\right)\right)$$

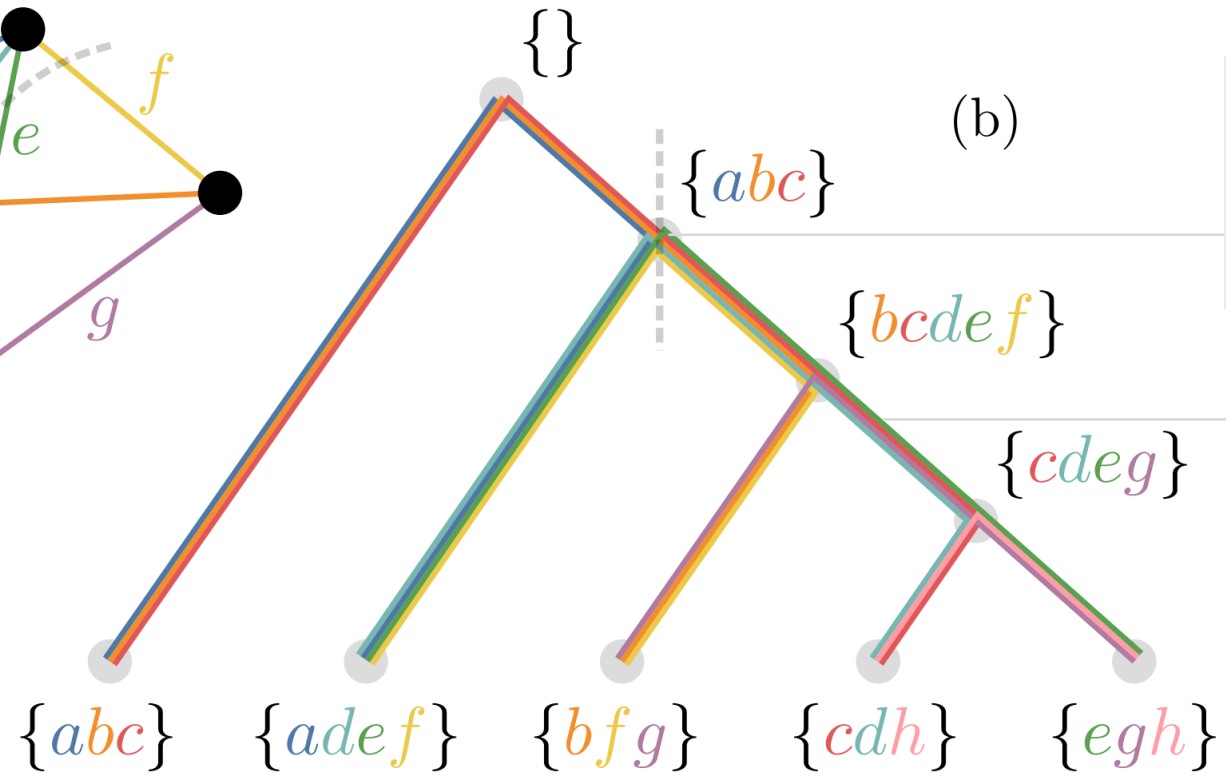
Each pairwise contraction is ~ matrix multiplication

… cost is extraordinarily sensitive to choice of order

Know optimal scaling is generically still exponential, in *treewidth of linegraph:* **Markov & Shi [2005]**

Contraction trees





vertex congestions give contraction flops – each associated with bipartition of graph G

edge congestions give intermediate tensor sizes – each associated with subgraph of G

Computational time and space given by *contraction tree:*

$$\left(\sum_{abc} T_0(x_{abc}) \left(\sum_{def} T_1(x_{adef}) \left(\sum_{g} T_2(x_{bfg}) \left(\sum_{h} T_3(x_{cdh}) T_4(x_{egh})\right)\right)\right)\right)$$

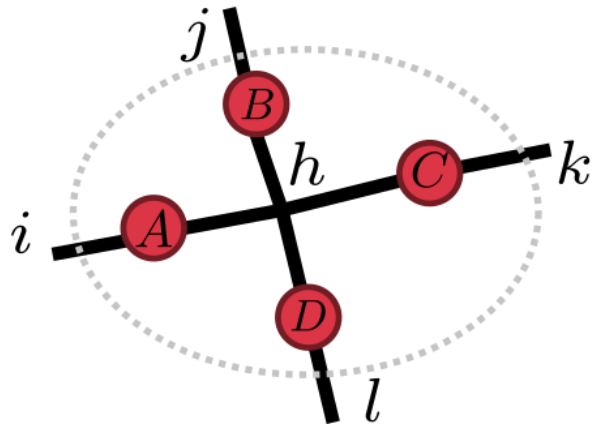
O'Gorman - Parameterization of tensor network contraction - 2019

Hypergraph Tensor Networks

Both sum-of-products defn and contraction tree permit indices appearing 3+ times, **hyper-edges**, AKA:

- Implicit COPY-tensor
- Z-spider
- linegraph of hyper graph still regular graph
- CANDECOMP/PARAFAC (CP) decomposition…

$$\psi_{ijkl} = \sum_h A_{ih} B_{jh} C_{kh} D_{lh}$$

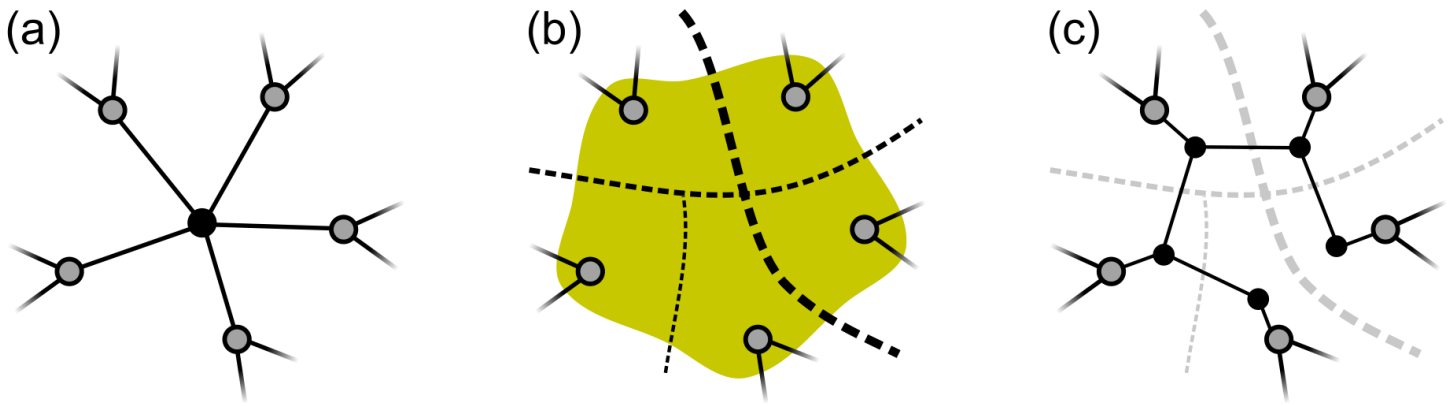


Contraction costs are now associated with hypergraph partitions, and pairwise contractions are performed like ~ batched-matrix-multiply

What's the benefit of hypergraphs?

Can resolve hyper-indices network built up of order-3 COPY-tensors…

But lose natural permutational symmetry of the problem **– best way to create decomposition now depends on contraction tree:**



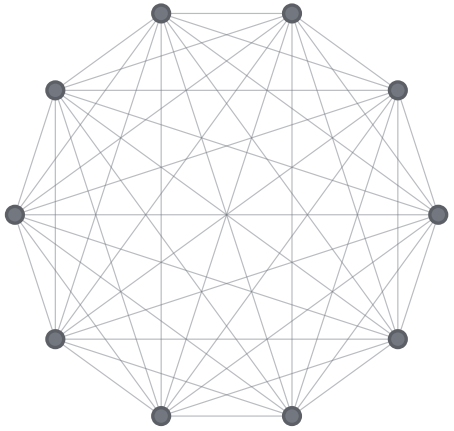
Hyper-edges have same cut-weight regardless of partition

Hypergraphs - an extreme all-to-all example

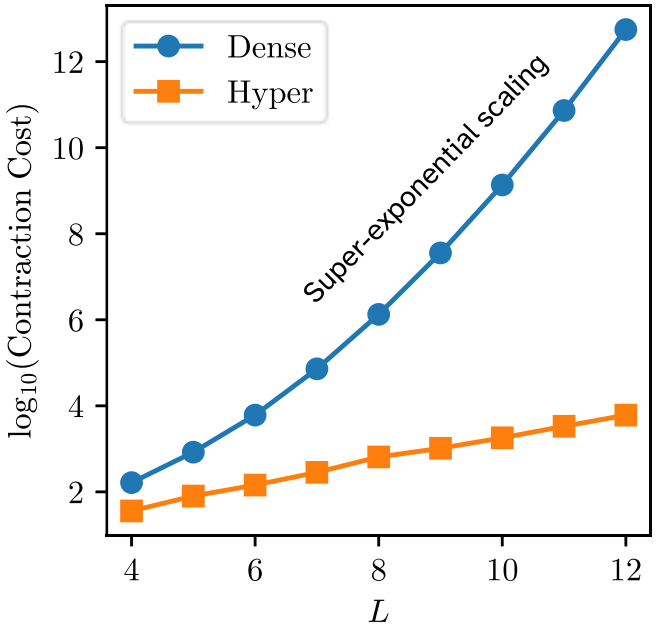
Take Ising model on graph with all-to-all interactions. Partition function given by contraction of tensor network where on each edge of graph place:

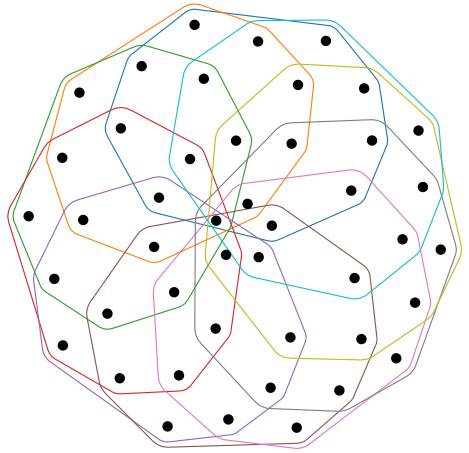
$$T_{i,j} = \begin{pmatrix} e^{-\beta J} & e^{\beta J} \\ e^{\beta J} & e^{-\beta J} \end{pmatrix}$$

with each (hyper)-index a site



Take $\sqrt{T}$ & absorb into vertices:





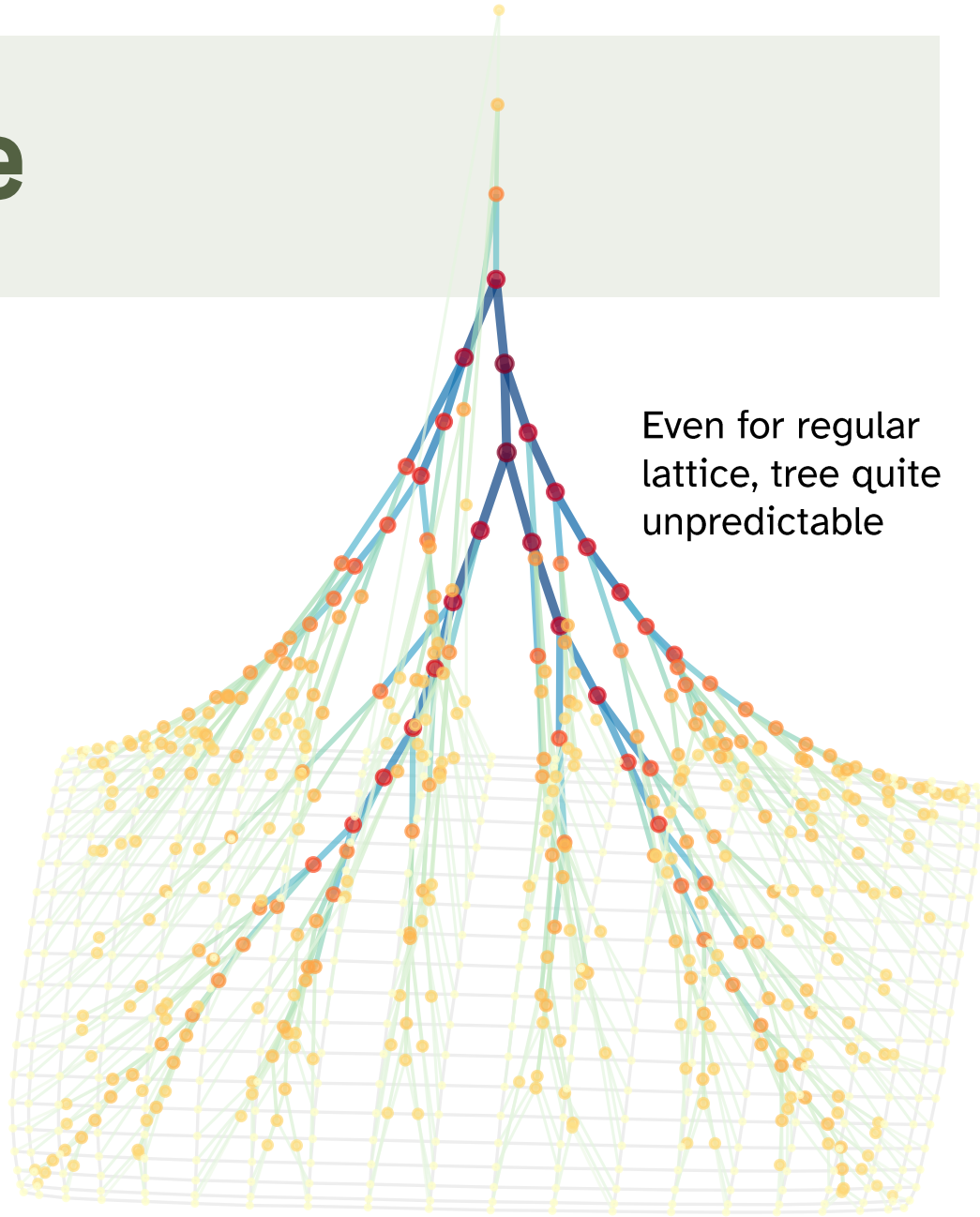
Simply leave as hypergraph:

Constructing the tree

arXiv:2002.01935 - use mix of:

1. Hypergraph-partitioning
2. Dynamic-programming
3. Bayesian optimization

To ‘learn’ how to construct and sample ever better trees for a particular geometry



Even for regular lattice, tree quite unpredictable

In all cases find better - often by orders of magnitude - performance than previous approaches… + seemingly close to optimal

Can we aid contraction by *simplifying* first?

Motivated by exact contraction, try and locally transform TN to:

1. Reduce number of tensors
2. Happily introduce hyperedges
3. Reduce weight of hypergraph cuts

Only expect gains for relatively structured TNs, but find in certain cases surprisingly powerful on their own...

Notable ‘fully simplifiable’ tensor networks

Matchgate Tensors:

- 3-planar-NOT-ALL-EQUAL (SAT or ‘ice’) and various problems [*Valiant* – Holographic Algorithms – 2004]
- Extended to non-planar [*Brayvi* - Contraction of matchgate tensor networks on non-planar graphs - 2008]
- …

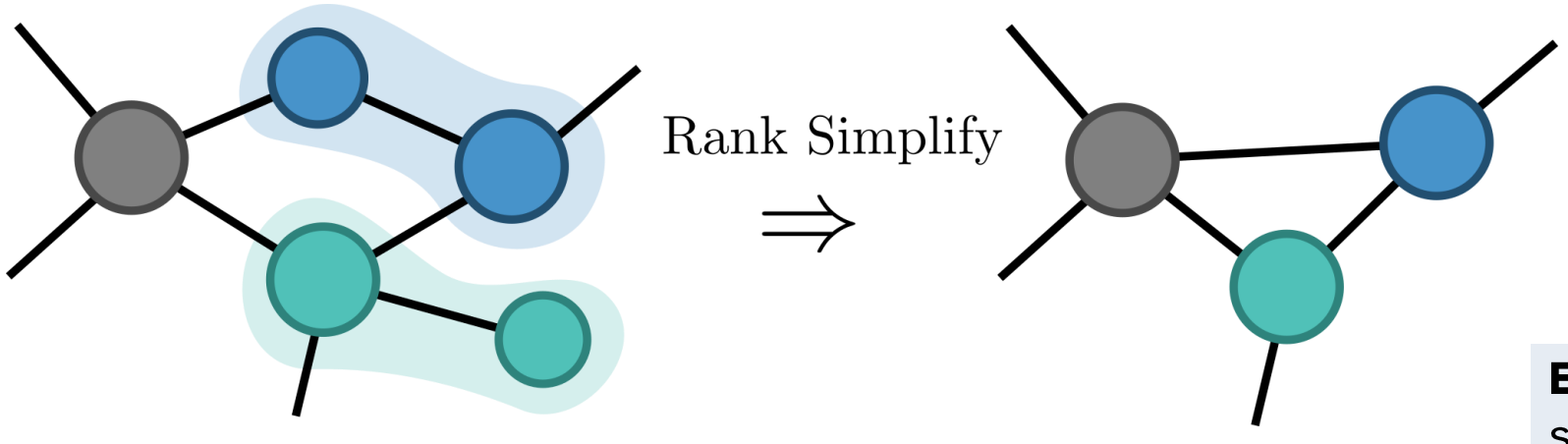
ZX-calculus:

- Clifford (stabilizer) qu. Circuits [van de Wetering - ZX-calculus for the working quantum computer scientist – 2020]
- XORSAT [*Beaudrap, Kissinger, Meichanetzidis* - Tensor Network Rewriting Strategies for Satisfiability and Counting – 2020]
- …

Local TN simplifications

Rank (#dim) Simplification

Perform any pairwise contraction that doesn't increase the rank of any tensor:



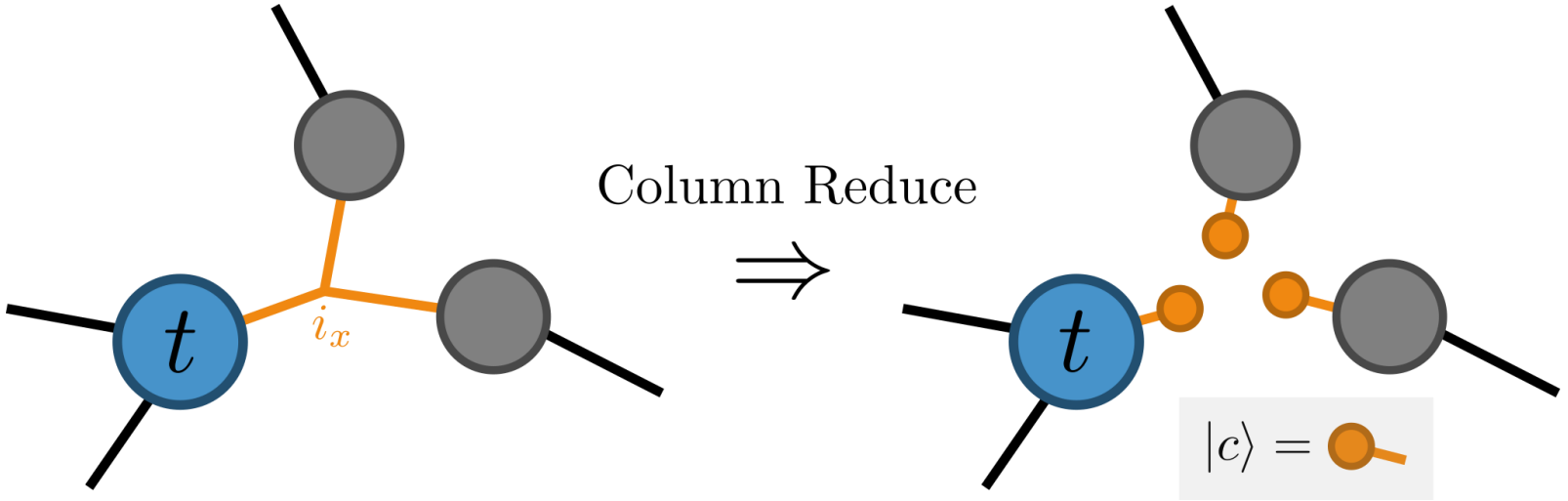
'Topology preserving' – all loops are retained.

Don't need to know anything about insides of tensors.

E.g. Qu. circuit simulation – can absorb any number of single qubit gates into neighbors

Column Reduction

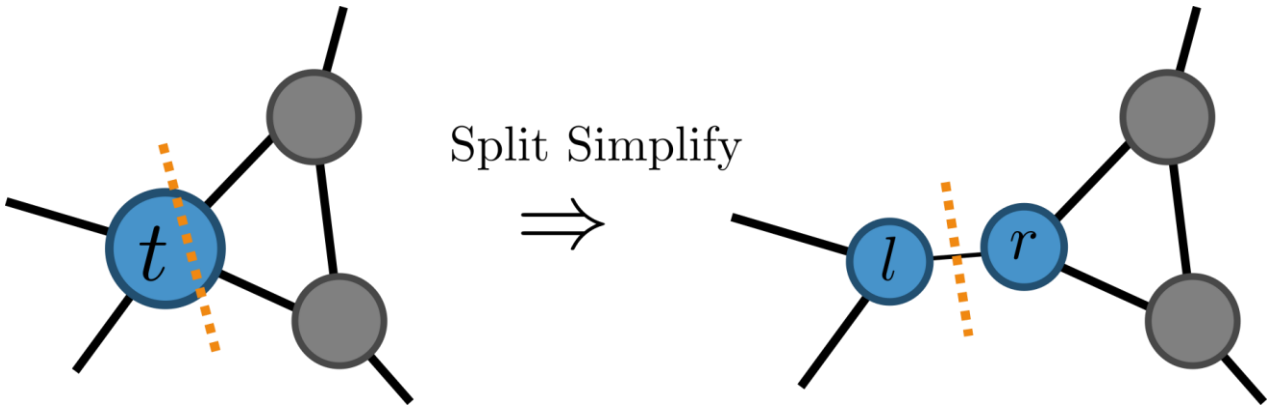
If any tensor has zero entries in all but one column – can remove that index from **all** tensors.



Mostly useful in conjunction with other simplification schemes.

Split Simplification

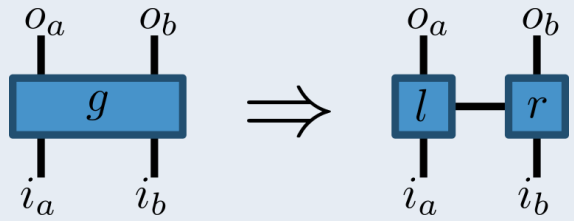
Search tensors for any low-rank (in terms of SVD) decompositions between all bipartitions of indices:



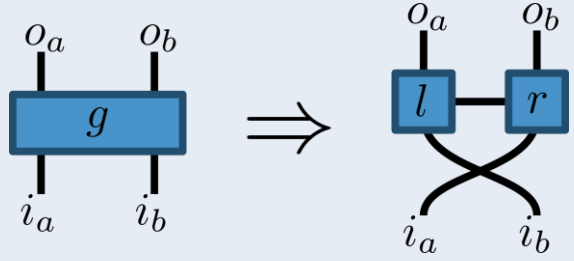
Creates more tensors… but *usually* lower weight cuts.

Examples

Any controlled gate, like CNOT:



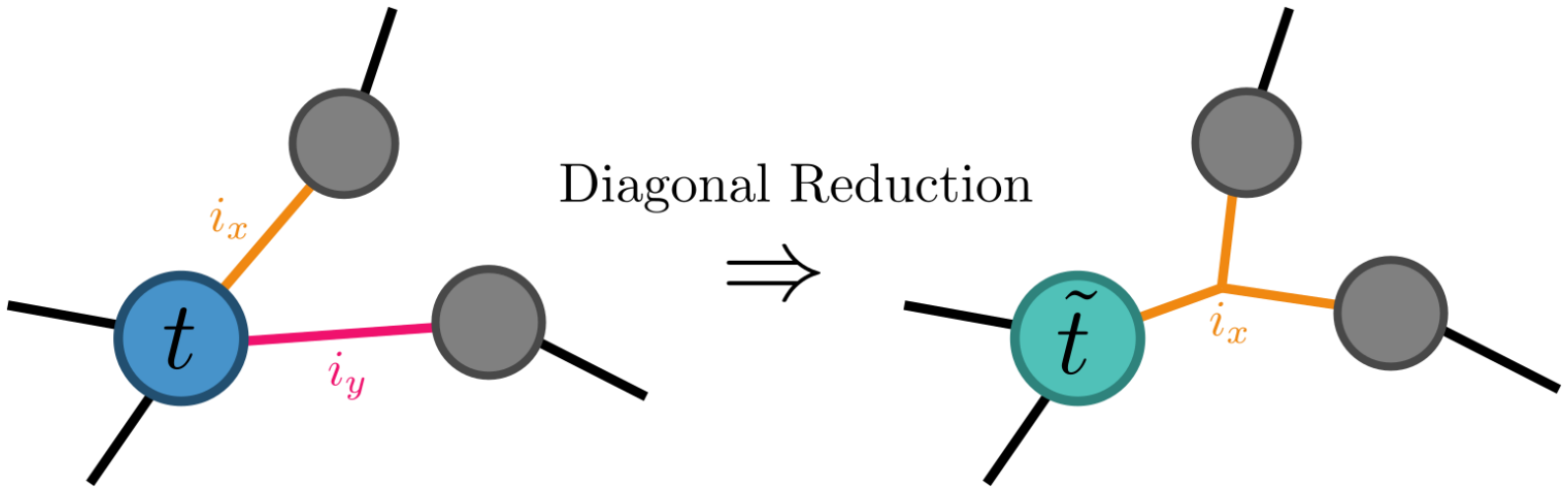
iSWAP:



Diagonal Reduction

Search tensors for any pairs of diagonal axes (i.e. zero whenever two indices are different):

$$T[\ldots, i_x, i_y] = T[\ldots, i_x, i_y]\delta_{i_x, i_y}$$



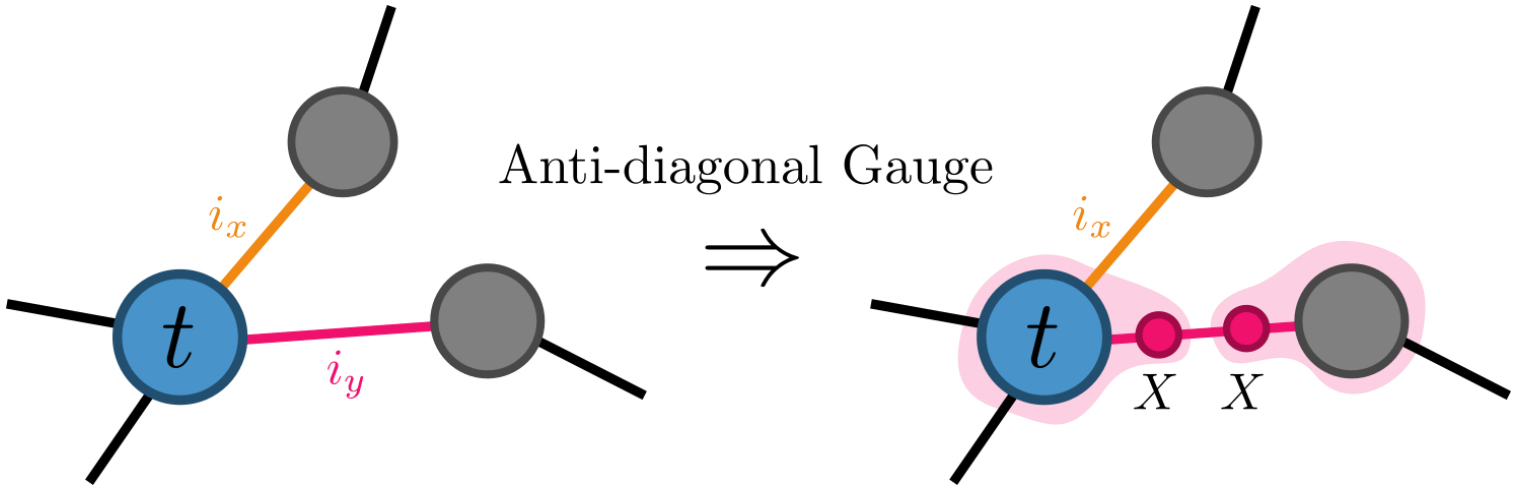
E.g. control qubit of gate after spatial decomposition

Can then take diagonal and replace index i_x with i_y everywhere...

introduction of hyper-index

Permutation Flipping

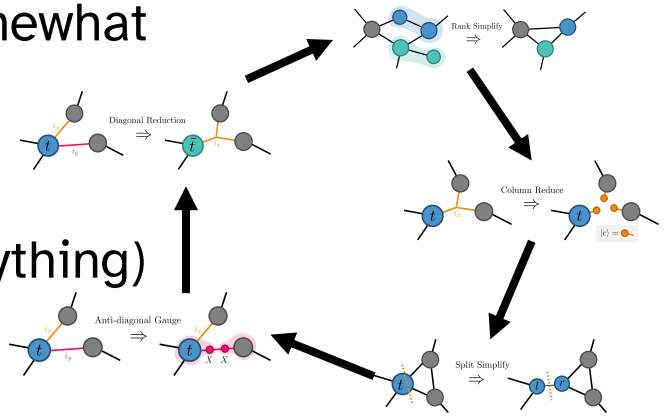
Can think about using gauge freedom of TNs to introduce any transformation along a bond that enables another simplification.



The simplest case is any index permutation that produces a diagonal pair of axes. (In $d = 2$ only such non-trivial gate is X). **Enables further diagonal reduction**

Combining simplifications

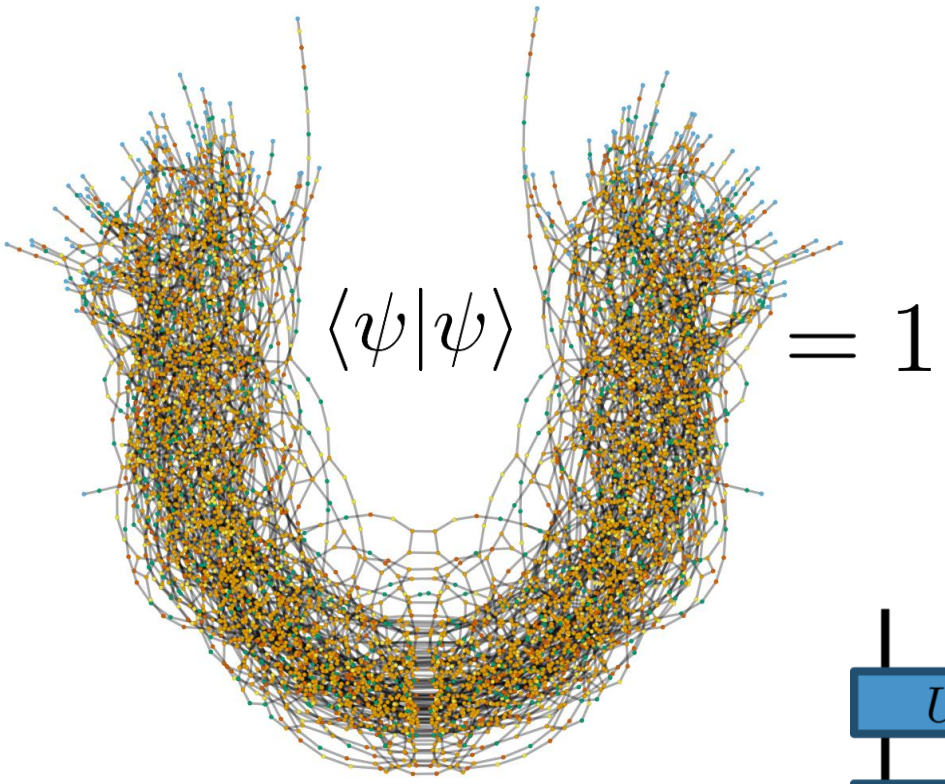
- Perform sequence of simplifications in loop until $|V|, |E|$ converges.
  - Process is pretty stable – but order of simplifications matters somewhat
- Need some numerical tolerance for most methods
  - Generally use something very small (not trying to approximate anything)



Whilst aim is to prepare TN for exact contraction, find huge reductions from the combination of these simple methods… sometimes!

Various results / applications

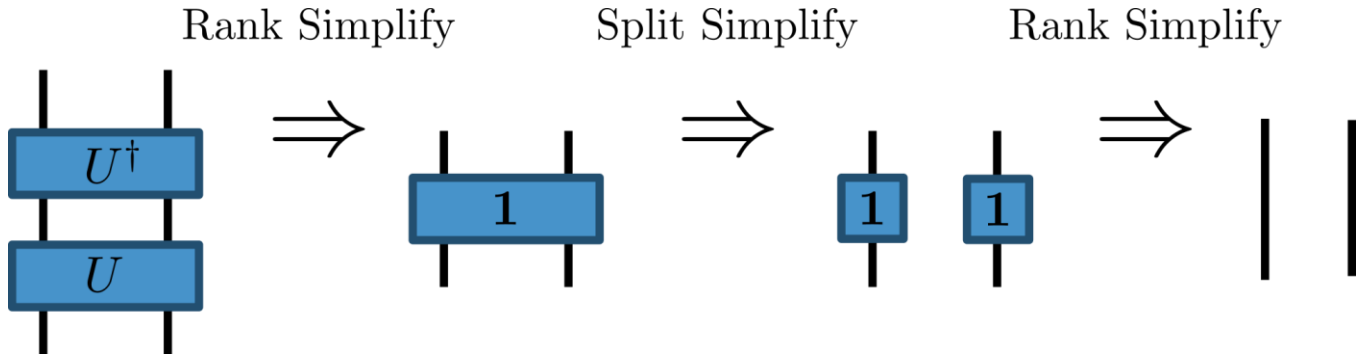
Fully simplifiable: norm of unitary TN



Deep, random, all-to-all circuit on ~100 qubits

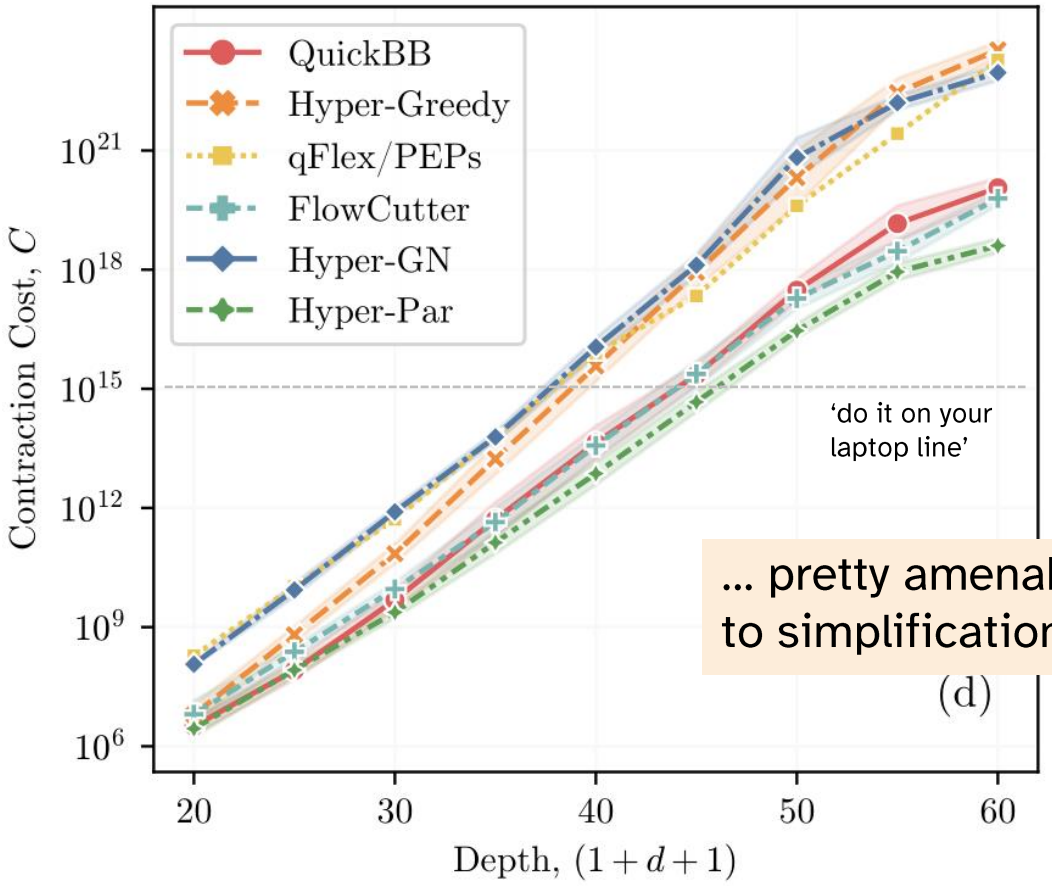
Lightcone structure of unitary tensor networks is automatically discovered + cancelled.

... not super useful as we know answer by construction – but illustrates how local simplification rules combine:



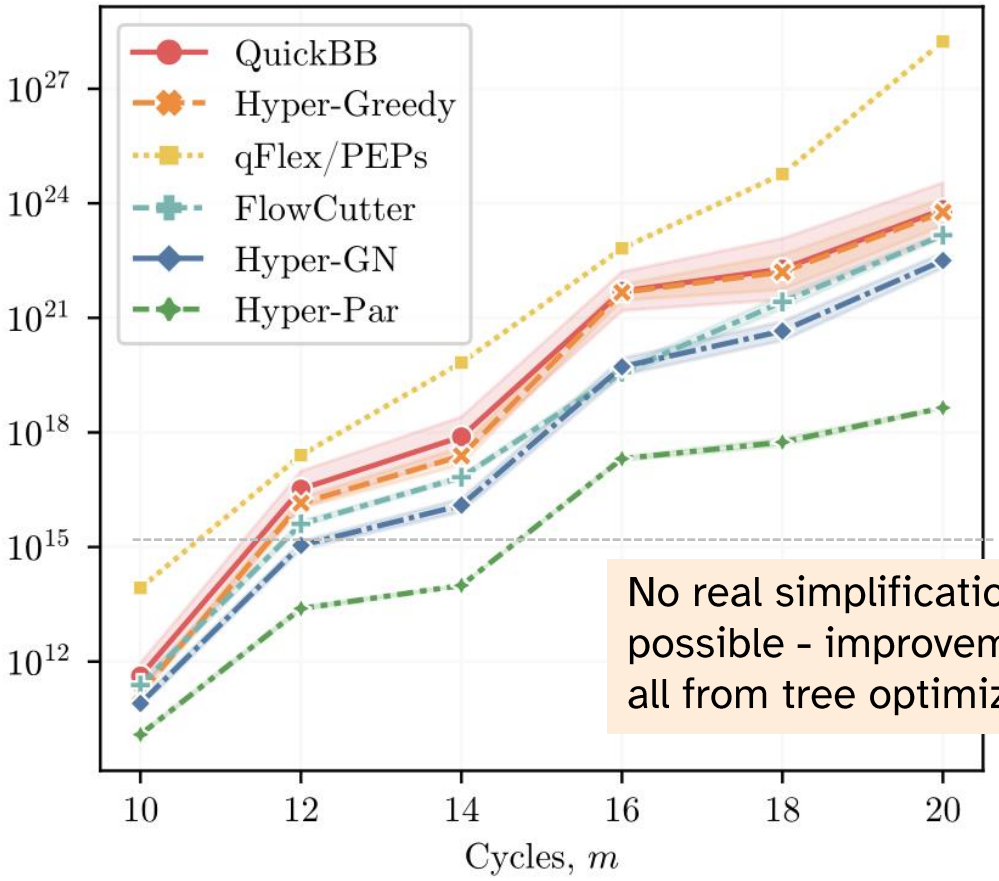
Random circuit amplitudes: rectangular & sycamore

Rectangular 7x7 quantum chip, **CZ** gates



... pretty amenable to simplifications

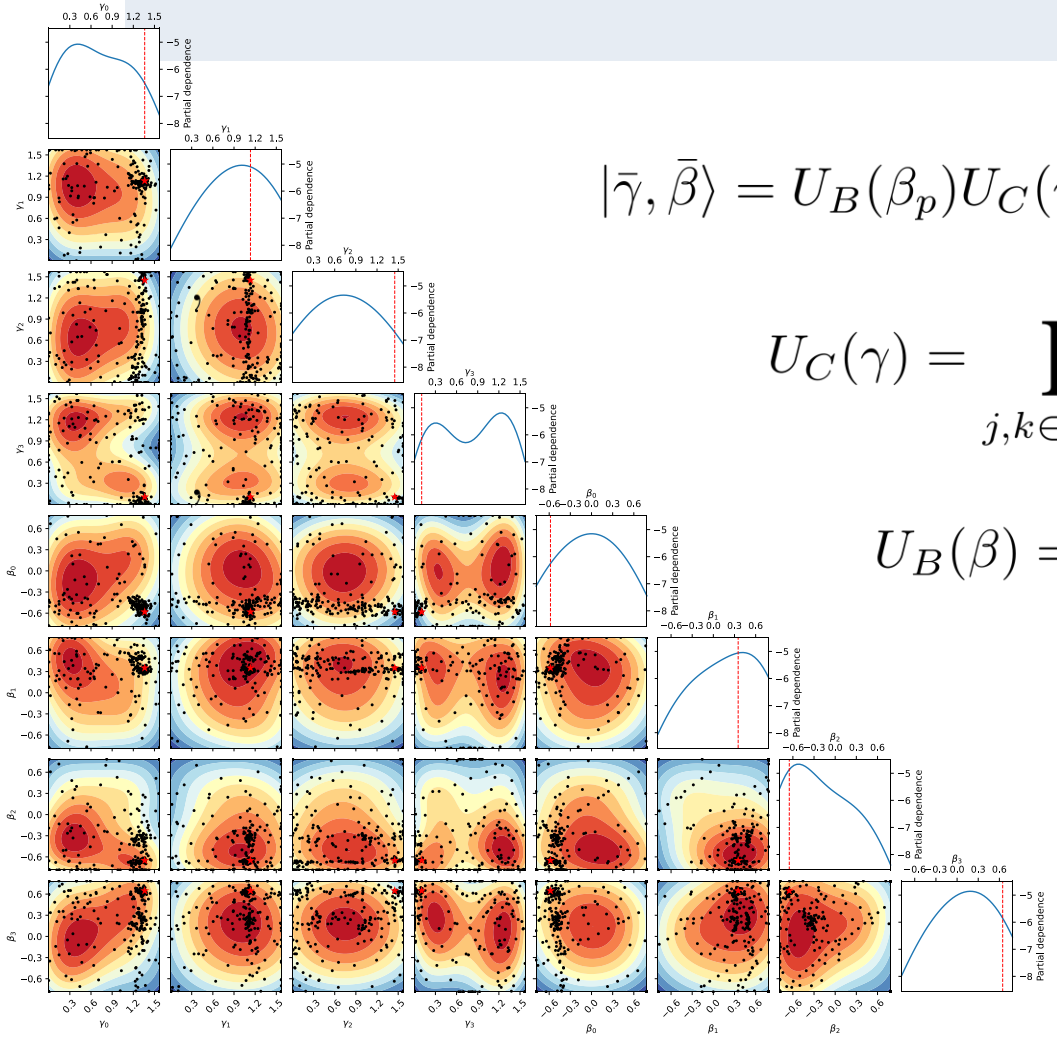
Sycamore – **FSIM** gates



No real simplifications possible - improvement all from tree optimizers

State-of-the-art performance that simulations of Google's Sycamore chips have built on: *arXiv:2005.06787, arXiv:2103.03074*

Quite simplifiable: QAOA circuit energy



Energy landscape

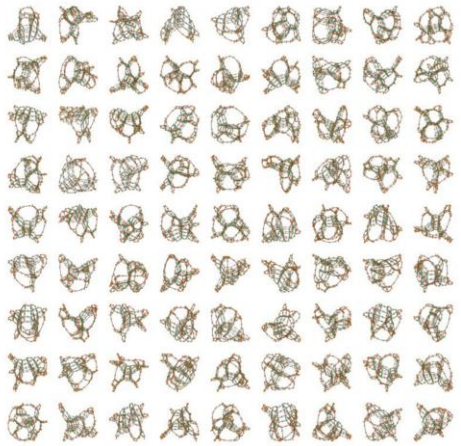
$$|\bar{\gamma},\bar{\beta}\rangle = U_B(\beta_p)U_C(\gamma_p)\cdots U_B(\beta_1)U_C(\gamma_1)|+\rangle$$

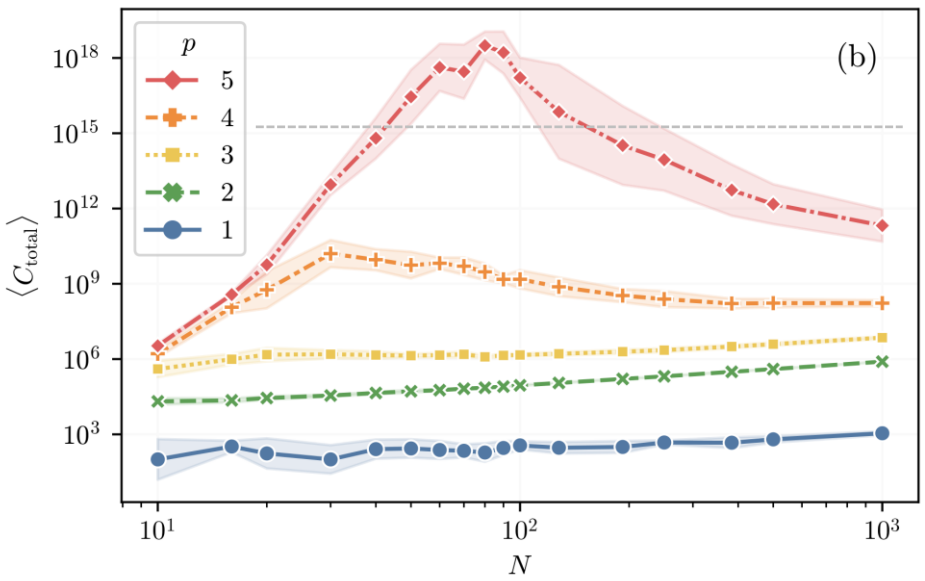
$$U_C(\gamma) = \prod_{j,k\in E(G)} e^{-i\gamma w_{jk} Z_j Z_k}$$

$$U_B(\beta) = \prod_{j\in G} e^{-i\beta X_j}$$

Easily assess N = 54 qubits $p = 4$ for random 3-regular graphs

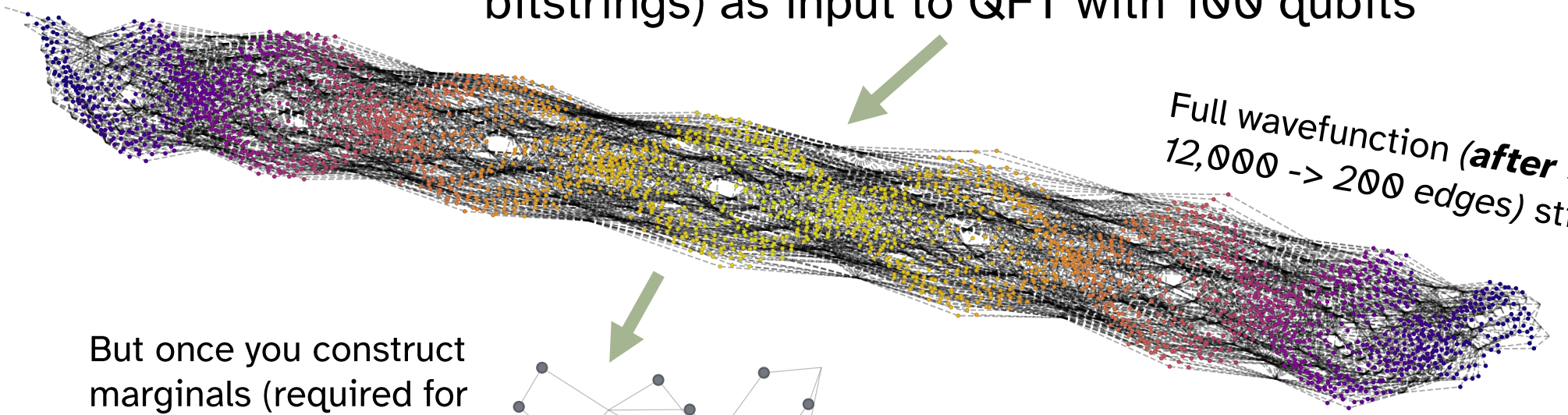
Local energy terms significantly simplify





Almost fully simplifiable: Quantum Fourier Transform

Take random product state (i.e. support on all input bitstrings) as input to QFT with 100 qubits



*Full wavefunction (**after** simplifications, 12,000 -> 200 edges) still very complex:*

But once you construct marginals (required for unbiased sampling), get for **all** conditional marginals of e.g. 5 qubits very simple TNs:

Some results about efficient TN QFT representations:

Scale invariance and efficient classical simulation of the quantum Fourier transform

Kieran J. Woolfe, Charles D. Hill, Lloyd C. L. Hollenberg

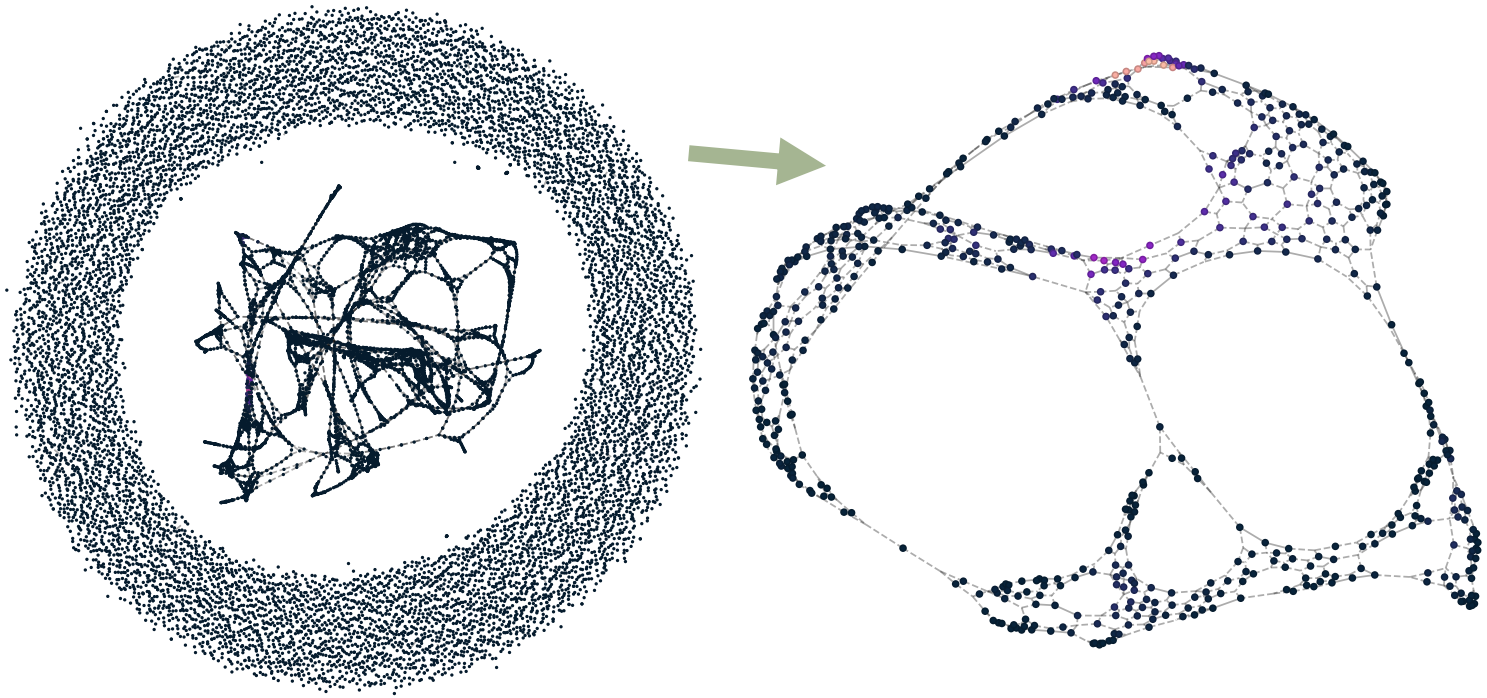
Non-quantum: (Weighted) Model Counting

$$x = \sum_{\{v\}} \prod_{v}^{\#\text{vars}} w_v \prod_{i}^{\#\text{clauses}} C_{\bar{v}_i}$$

Very naturally described by hyper tensor network:

- Single variable is hyperindex
- OR-clause tensors e.g. can be CP decomposed

Of 100 instances from the 2020 Model Counting Competition can solve 99 (compared to 69 for next best exact weighted model counter, ADDMC).



Essentially all simplify to either a trivial or small core hyper tensor network which can be easily contracted:

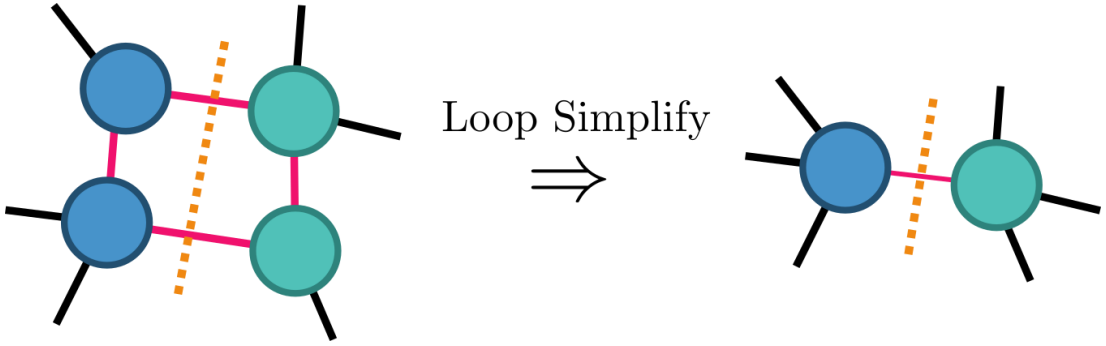
Approximate contraction?

- Know many TNs can be *approximately* contracted (although don't expect all)
  - In fact, for many-body TN usages, nearly always approximately contract

Approximate contraction?

Approx. simplifications:

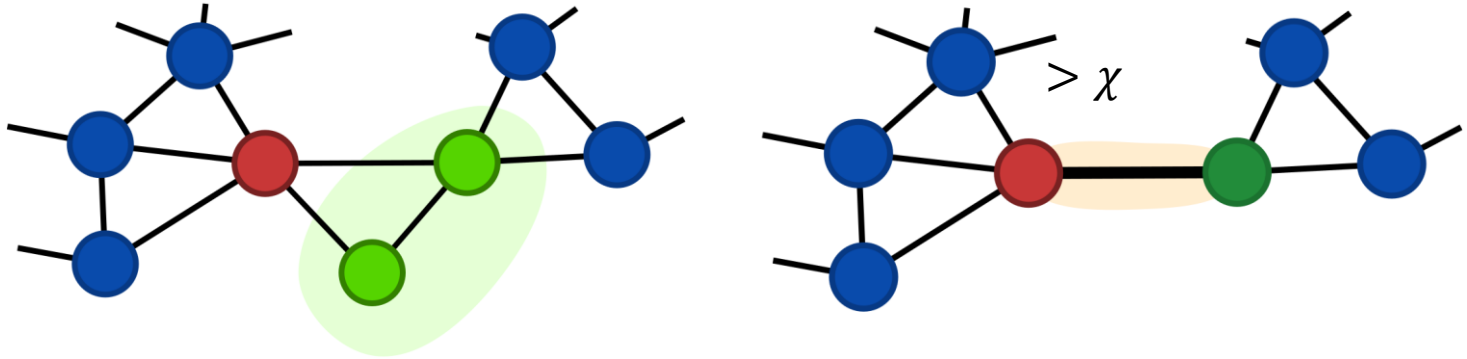
E.g. loop simplification and other more non-local decompositions



Approx. contractions:

Augment contraction tree with compressions, need to consider:

- Gauging
- Estimating error
- … many more subtleties

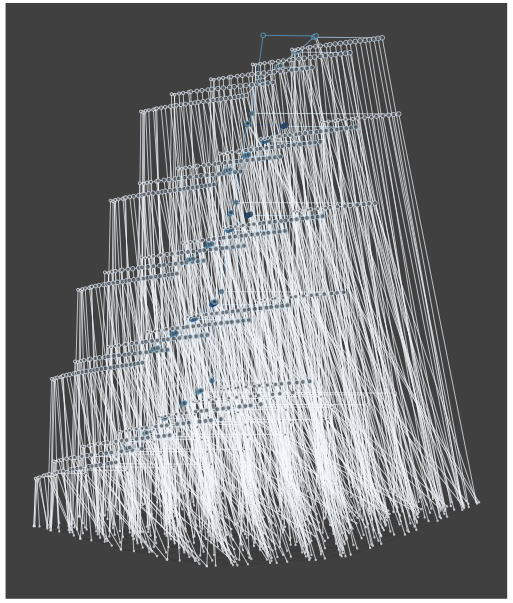


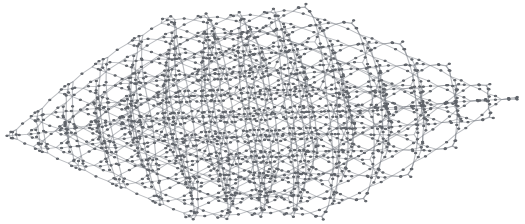
Contract according to series of merges (the tree)

Compress whenever shared bonds grow beyond χ

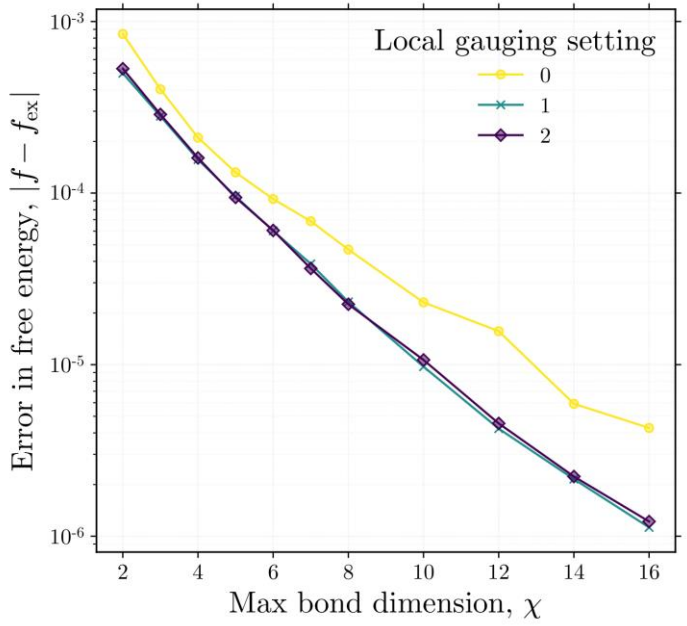
Interesting questions for approx. contraction:

Contract useful things where geometry is 'just another input'

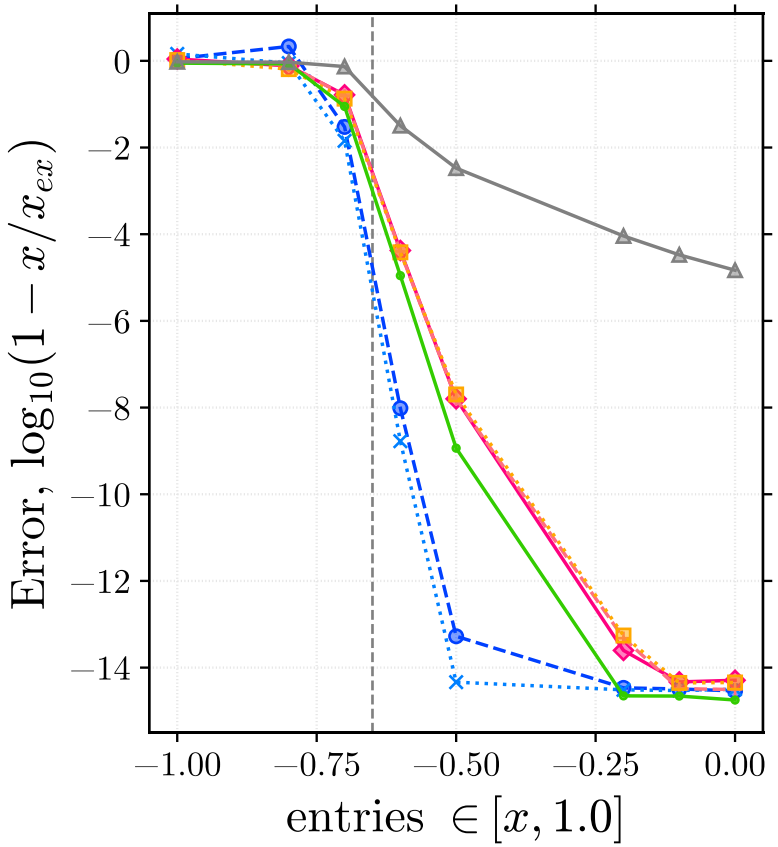




Frustrated pyrochlore lattice



Investigate the line between approx. hard and easy:



Tuning distribution random tensor entries are drawn from, find transition like behavior

Concluding thoughts

- Still finding places where extended exact contraction capabilities might be useful
- What problems can we extend approximate contraction to?
- How can we mix contraction + simplification + approximation simultaneously?
- Implementations available: github.com/jcmgray/quimb + github.com/jcmgray/cotengra 
- Thanks!