Title: Quantum computation with Topological Lattice Field Theories

Date: Feb 04, 2009 04:00 PM

URL: http://pirsa.org/09020026

Abstract: Recent results have shown that quantum computers can approximate the value of a tensor network efficiently. These results have initiated a search for tensor networks which contract to computationally interesting quantities. Topological Lattice Field Theories (TLFTs) are one source of such networks; when defined appropriately, networks arising from TLFTs contract to give topological invariants. In this elementary talk, we will define and classify TLFTs which lead to invariants of surfaces, and sketch out the corresponding quantum algorithm. Our exposition will be targetted at a general mathematically-inclined audience; no previous knowledge of field theories is required.

A LEISURELY STROLL THROUGH TLFTs*

(and how they may be useful in quantum computation)

Gorjan Alagic
IQC / C&O U.Waterloo

Other people thinking about this stuff include:

Alexander Russell (Uconn)
Cris Moore (UNM/SFI)
Jon Yard (LANL)
Zeph Landau (CUNY)
... and others.

*Topological Lattice Field Theories, of course!

A LEISURELY STROLL THROUGH TLFTs*

(and how they may be useful in quantum computation)

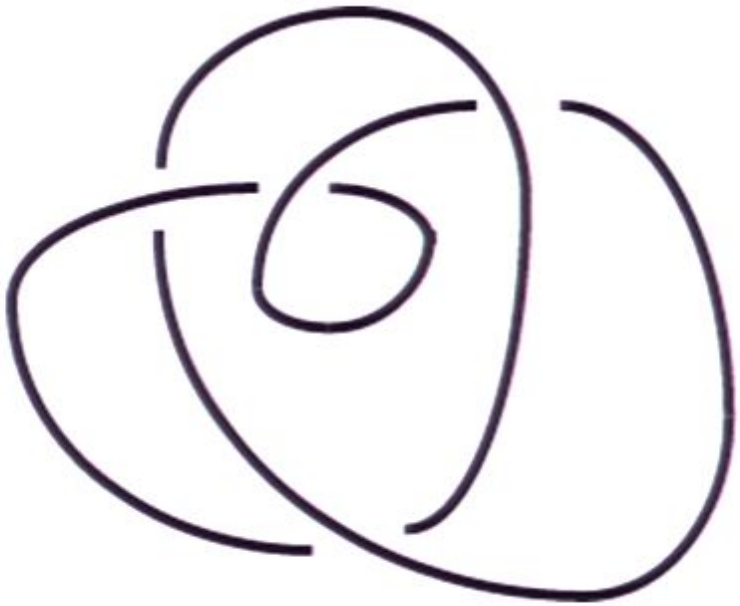
Gorjan Alagic
IQC / C&O U.Waterloo

Other people thinking about this stuff include:

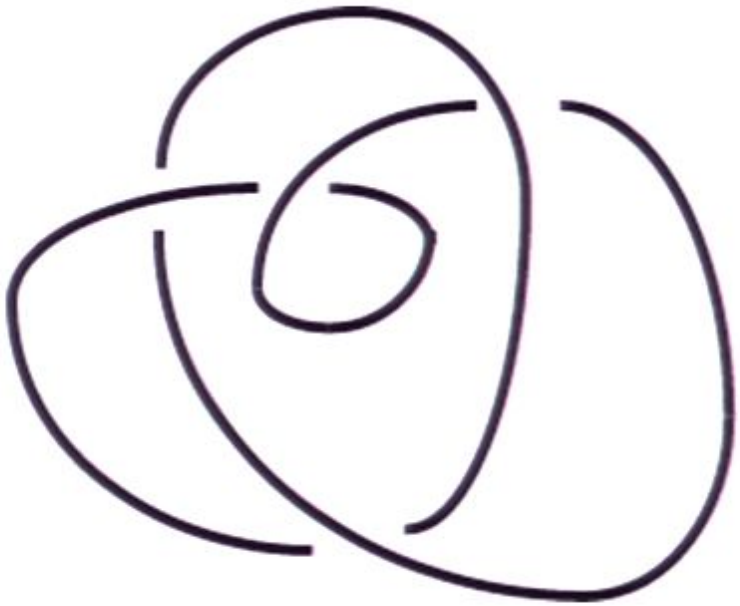
Alexander Russell (Uconn)
Cris Moore (UNM/SFI)
Jon Yard (LANL)
Zeph Landau (CUNY)
... and others.

*Topological Lattice Field Theories, of course!

Knots

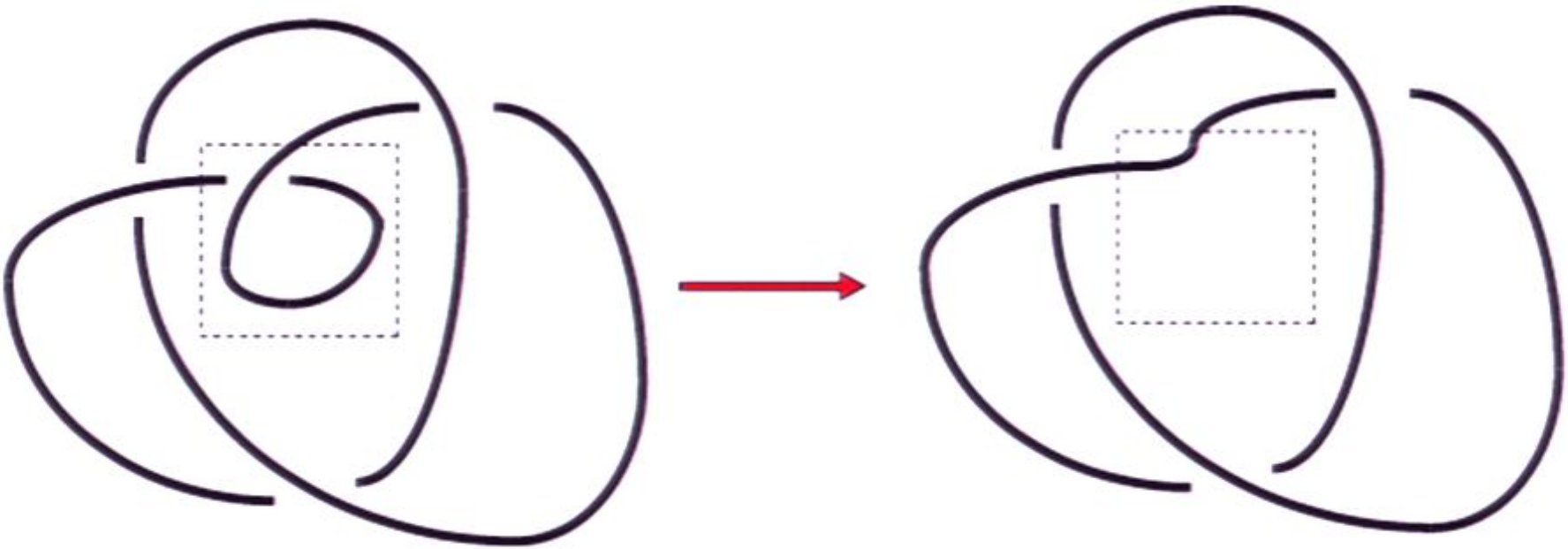


Knot diagrams

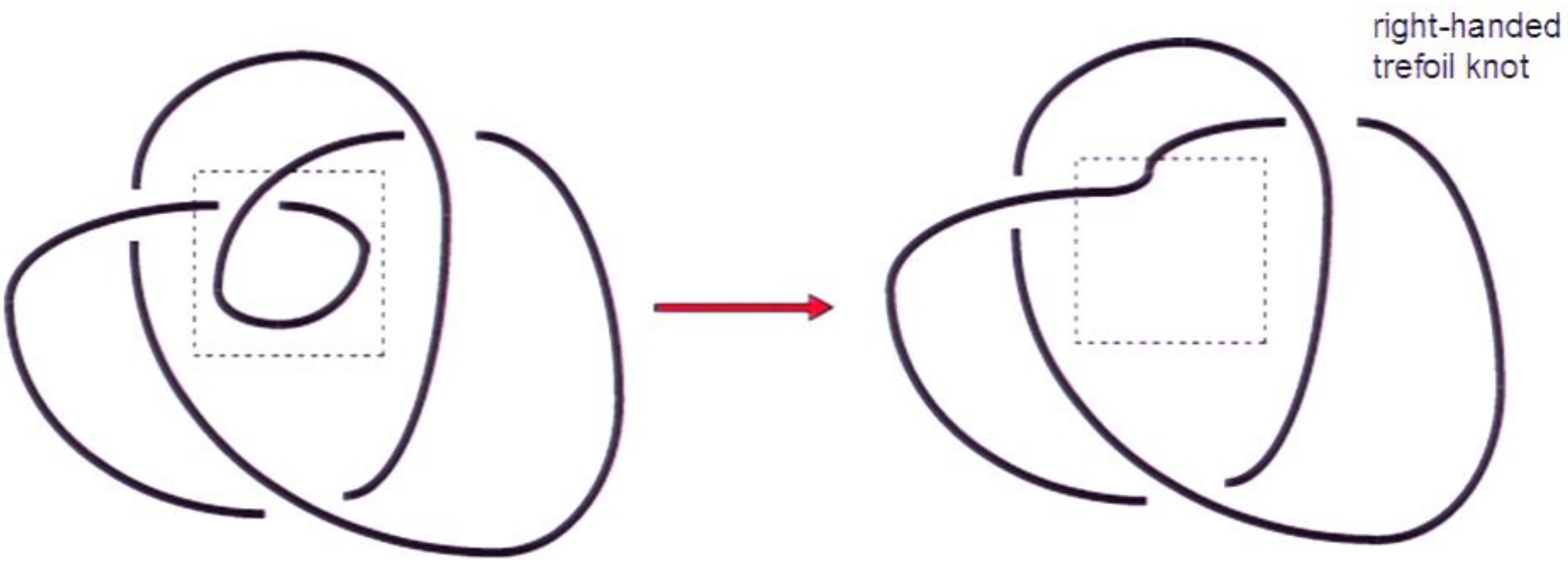


Knot and link diagrams

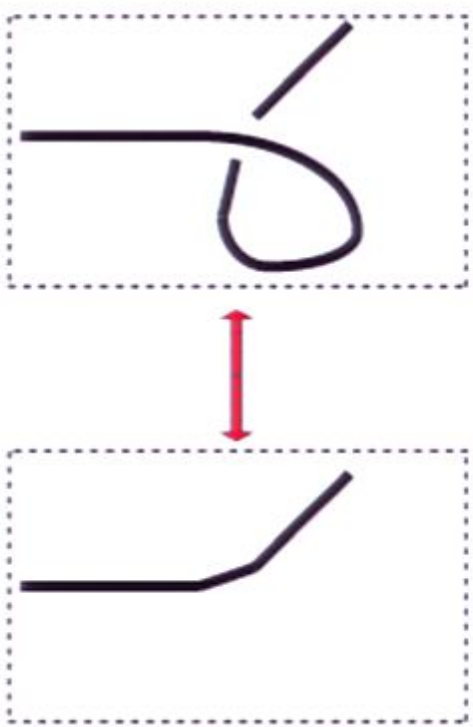
Knot and link diagrams



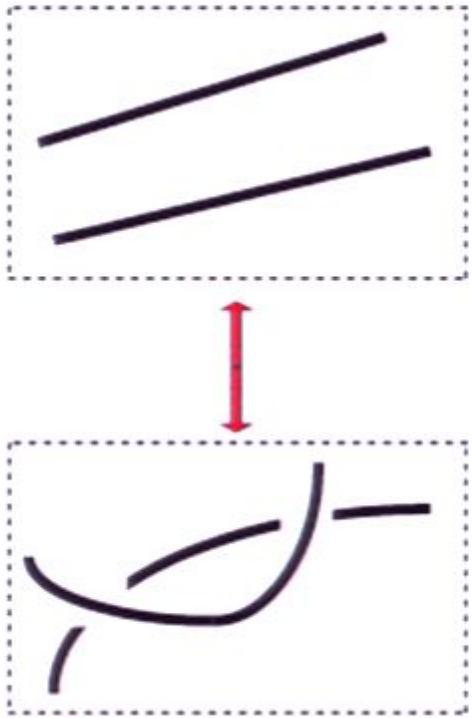
Knot and link diagrams



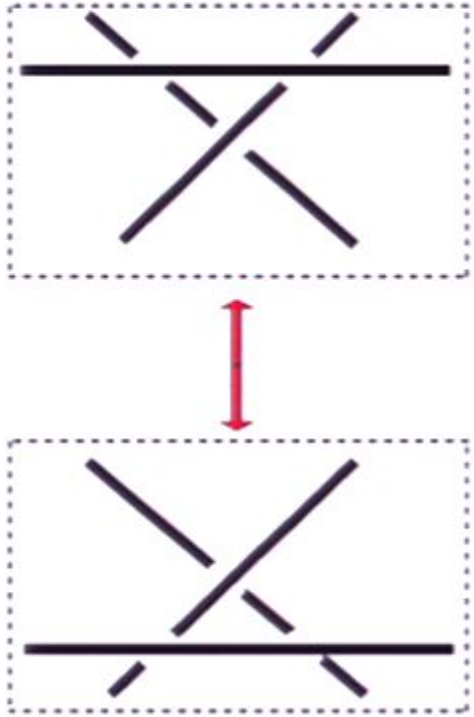
Reidemeister Moves



Type I



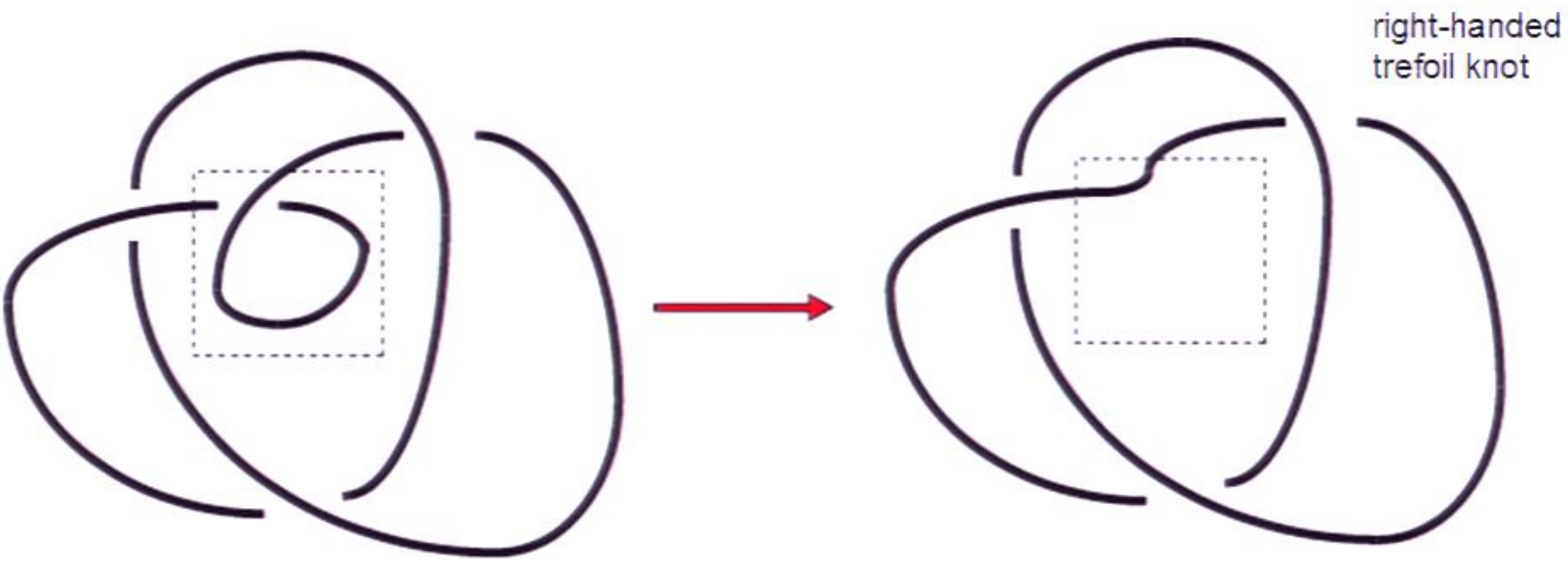
Type II



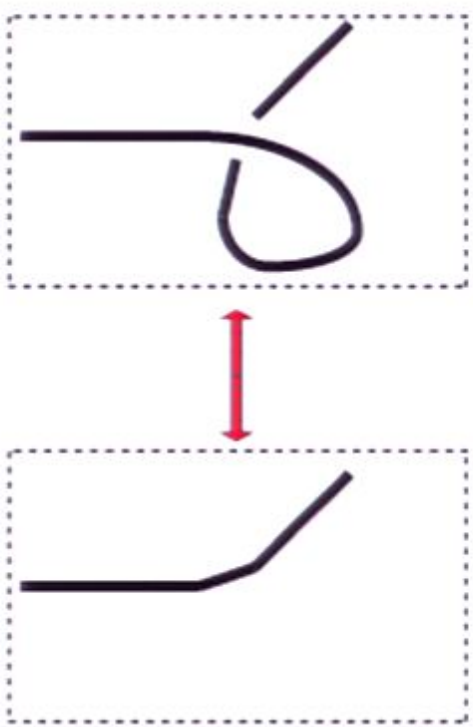
Type III

Purely combinatorial rules!

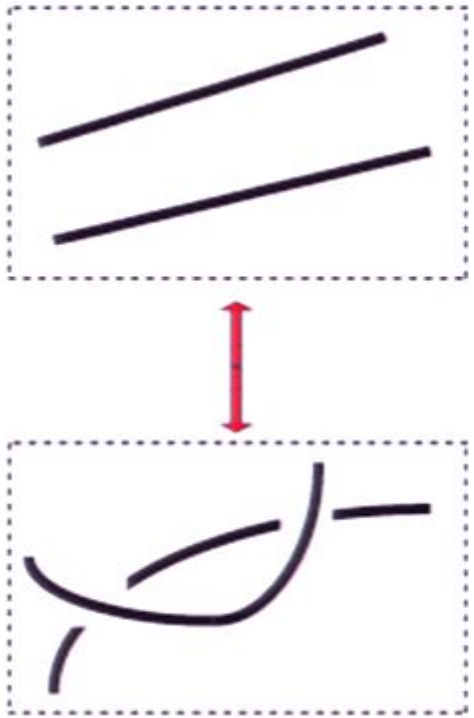
Knot and link diagrams



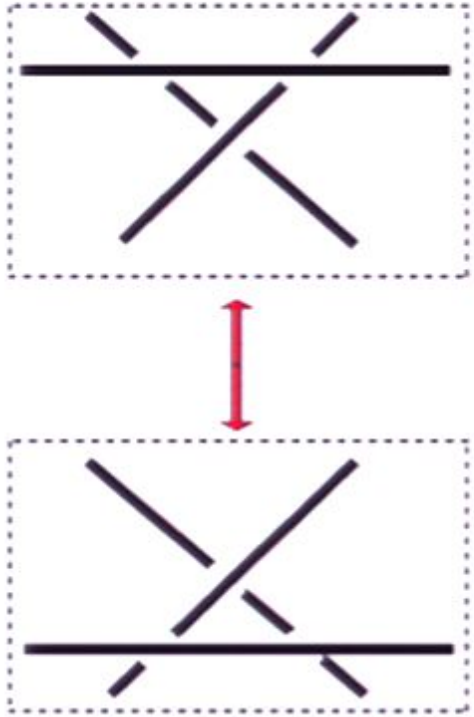
Reidemeister Moves



Type I



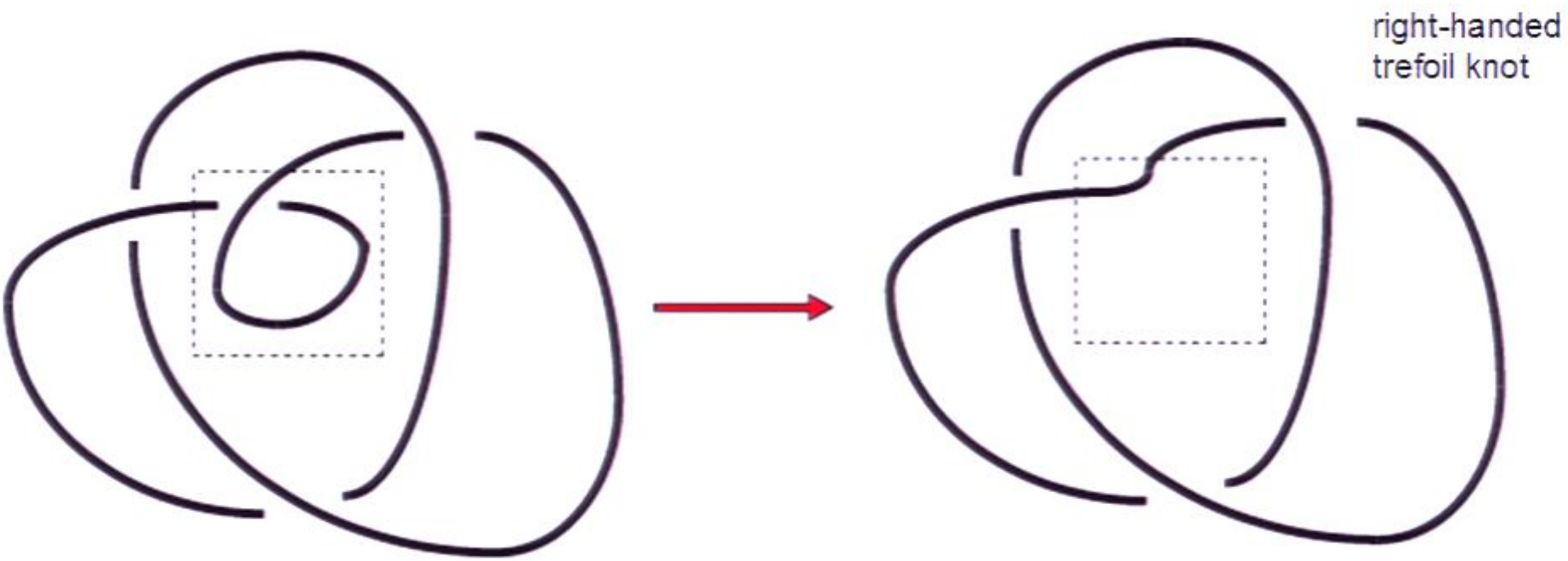
Type II



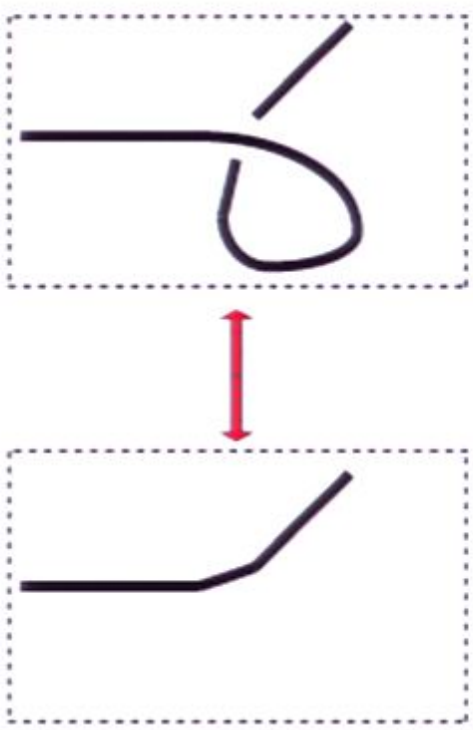
Type III

Purely combinatorial rules!

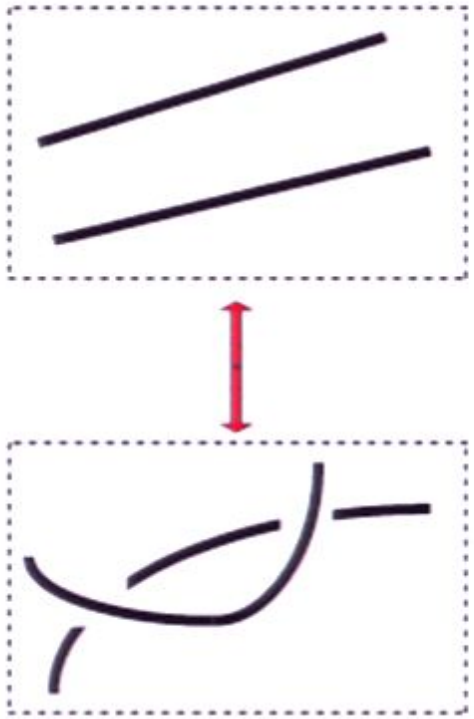
Knot and link diagrams



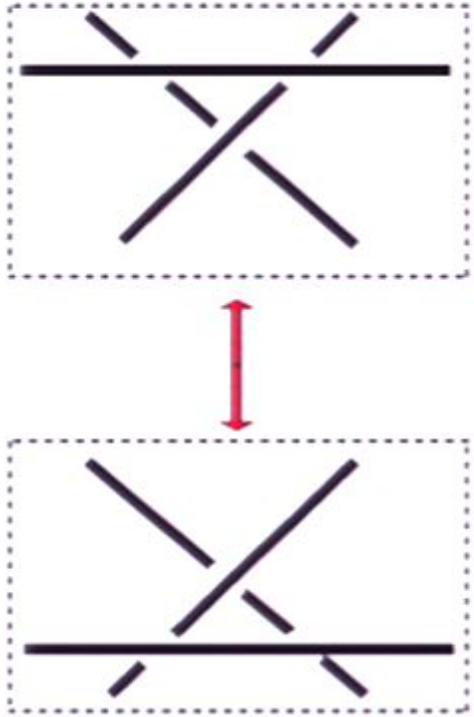
Reidemeister Moves



Type I



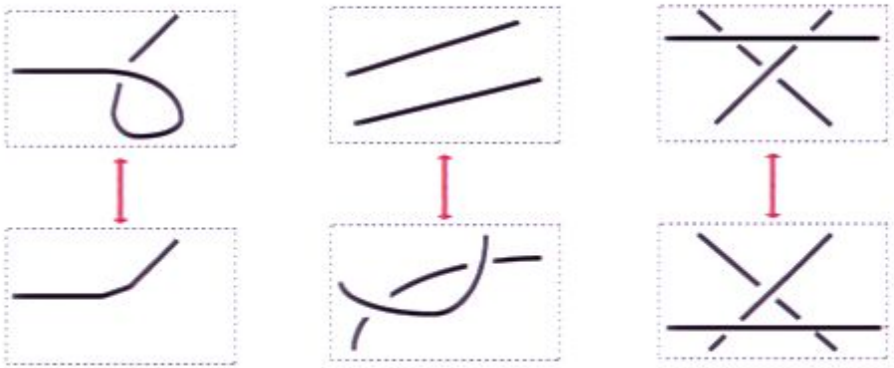
Type II



Type III

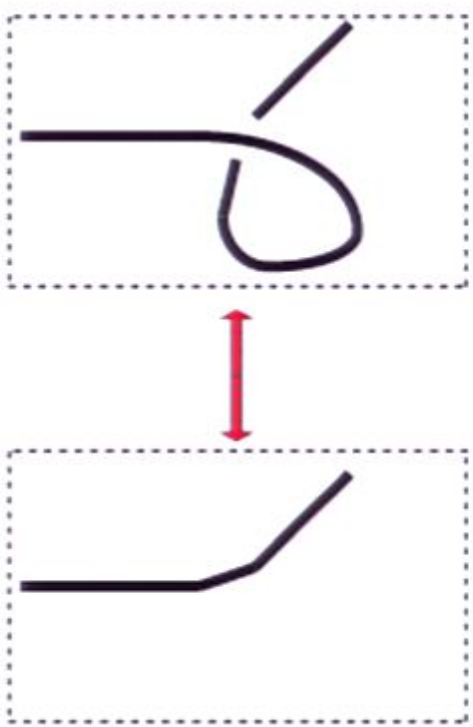
Purely combinatorial rules!

Reidemeister Moves

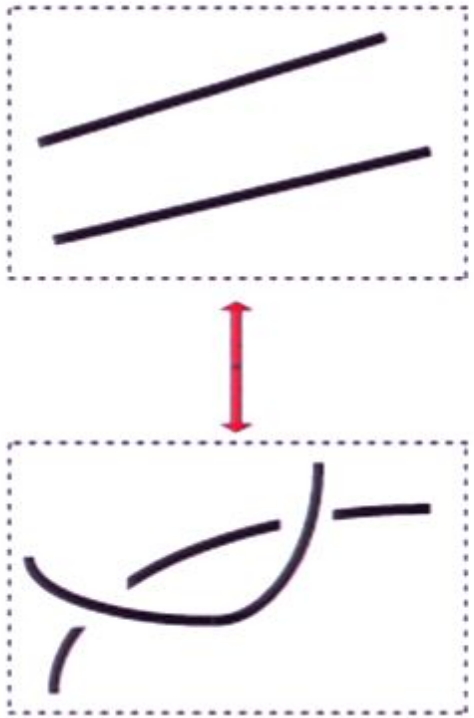


Theorem (Alexander and Briggs, Reidemeister) [1927]:
Any two diagrams of the same link differ by a finite number of these moves.

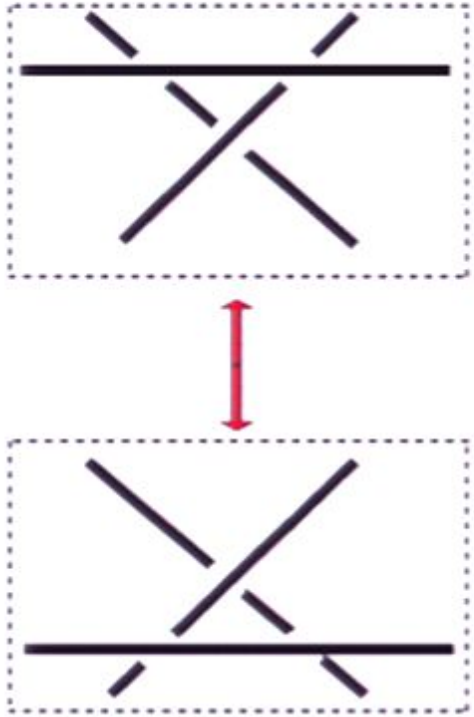
Reidemeister Moves



Type I



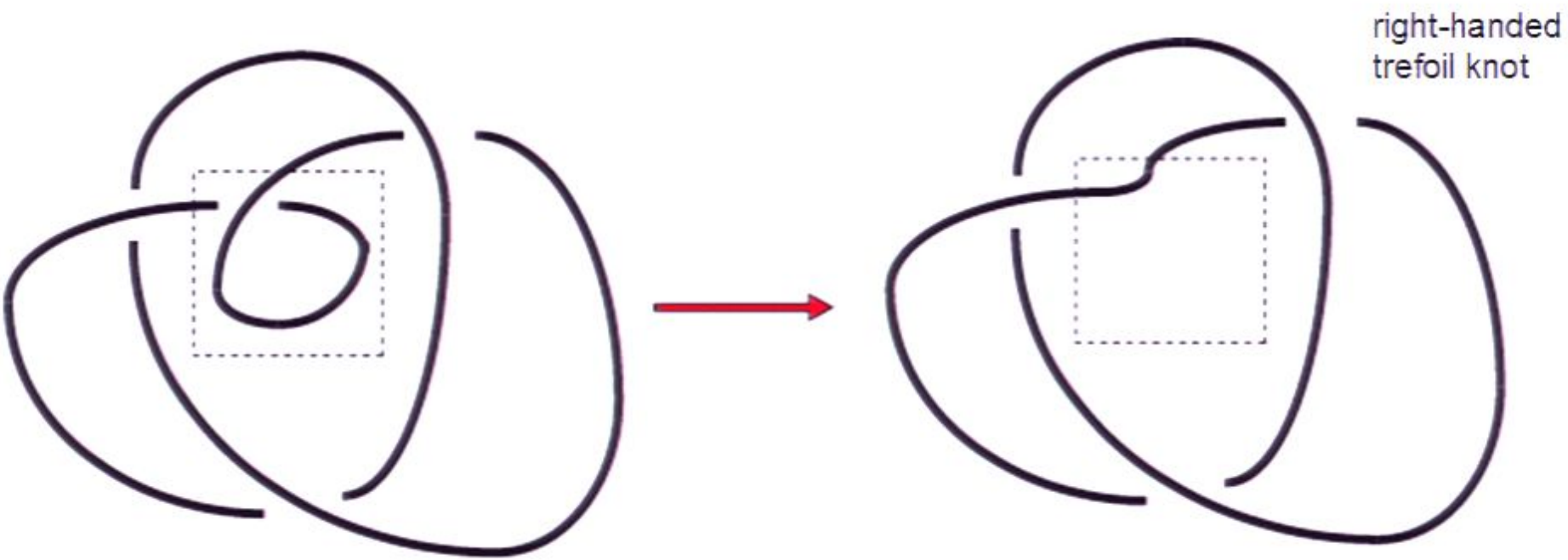
Type II



Type III

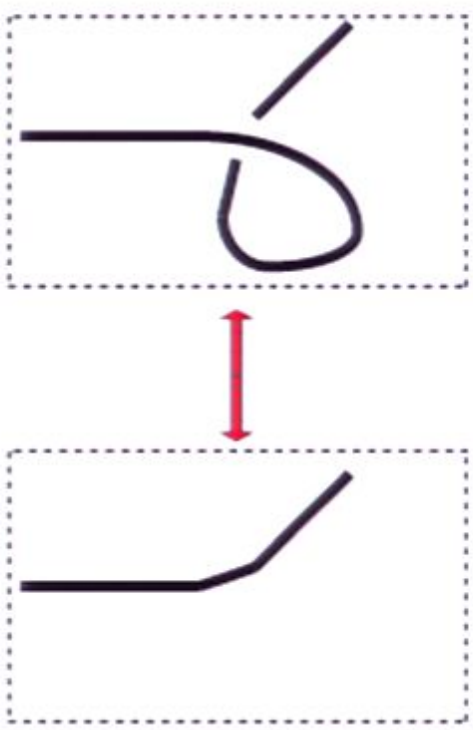
Purely combinatorial rules!

Knot and link diagrams

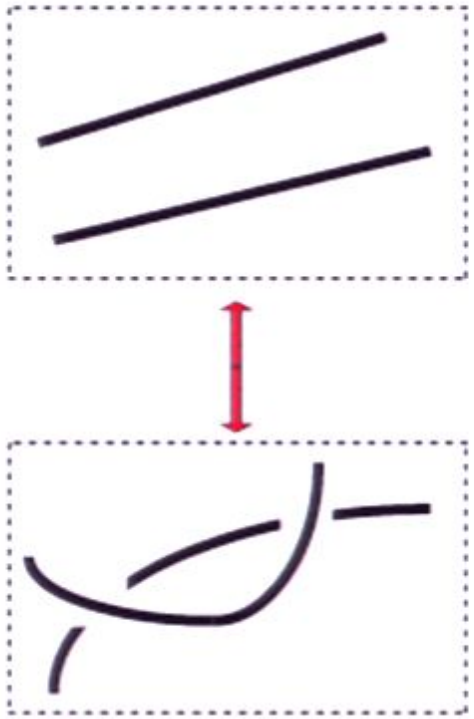


"Type I Reidemeister Move"

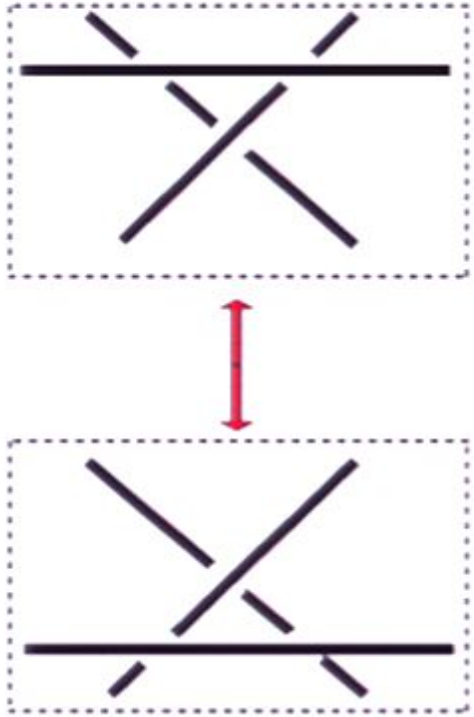
Reidemeister Moves



Type I



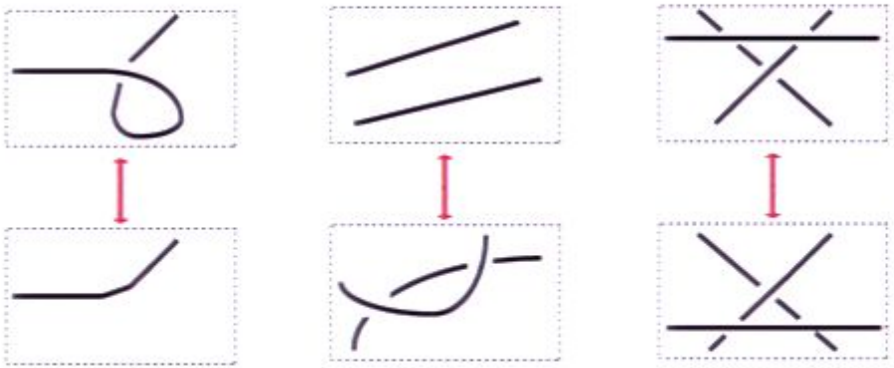
Type II



Type III

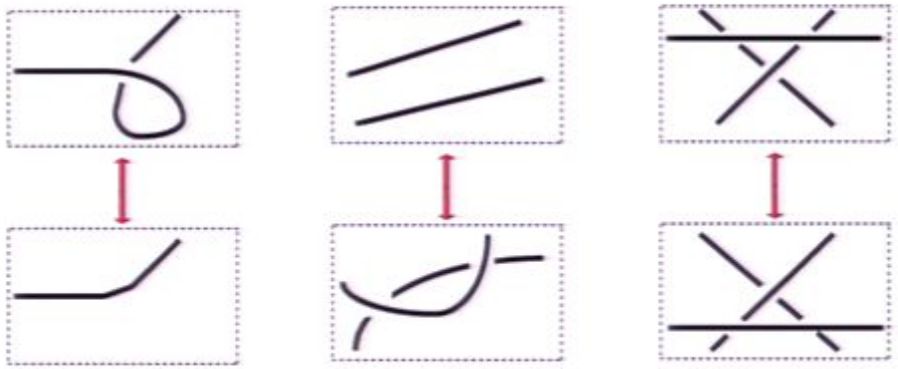
Purely combinatorial rules!

Reidemeister Moves



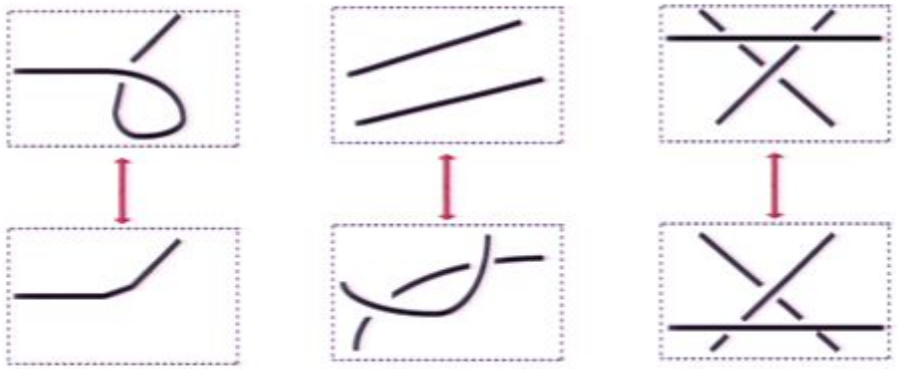
Theorem (Alexander and Briggs, Reidemeister) [1927]:
Any two diagrams of the same link differ by a finite number of these moves.

Reidemeister Moves



Theorem (Alexander and Briggs, Reidemeister) [1927]:
Any two diagrams of the same link differ by a finite number of these moves.

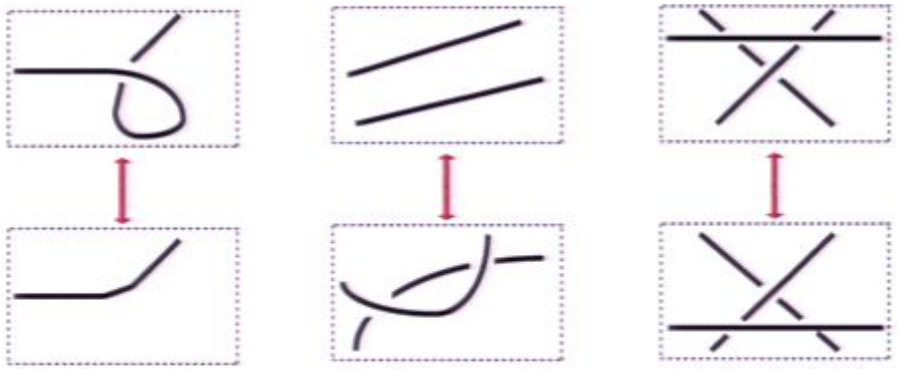
Reidemeister Moves



Theorem (Alexander and Briggs, Reidemeister) [1927]:
Any two diagrams of the same link differ by a finite number of these moves.

This is great!
Does something analogous happen in higher dimensions?

Reidemeister Moves



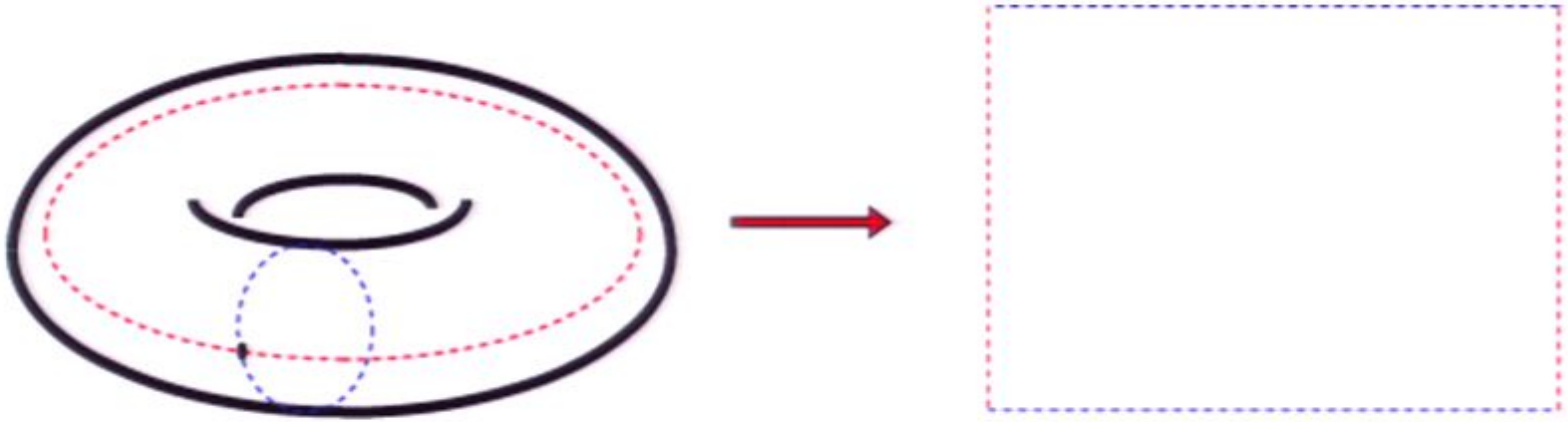
Type I Type II Type III

Theorem (Alexander and Briggs, Reidemeister) [1927]:
Any two diagrams of the same link differ by a finite number of these moves.

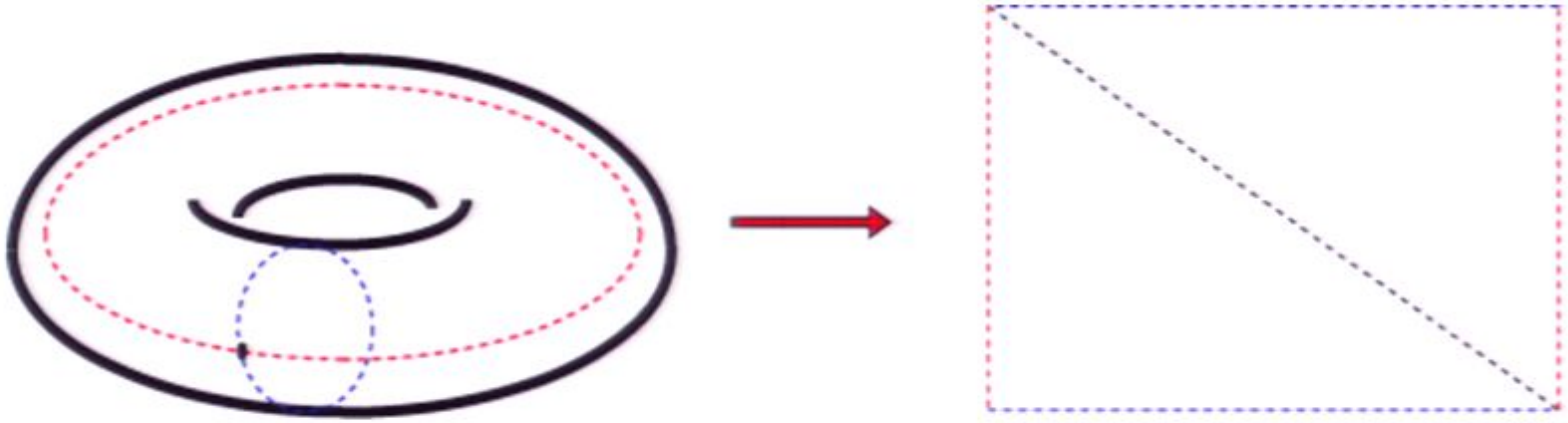
This is great!
Does something analogous happen in higher dimensions?

Orientable Surfaces

Orientable Surfaces



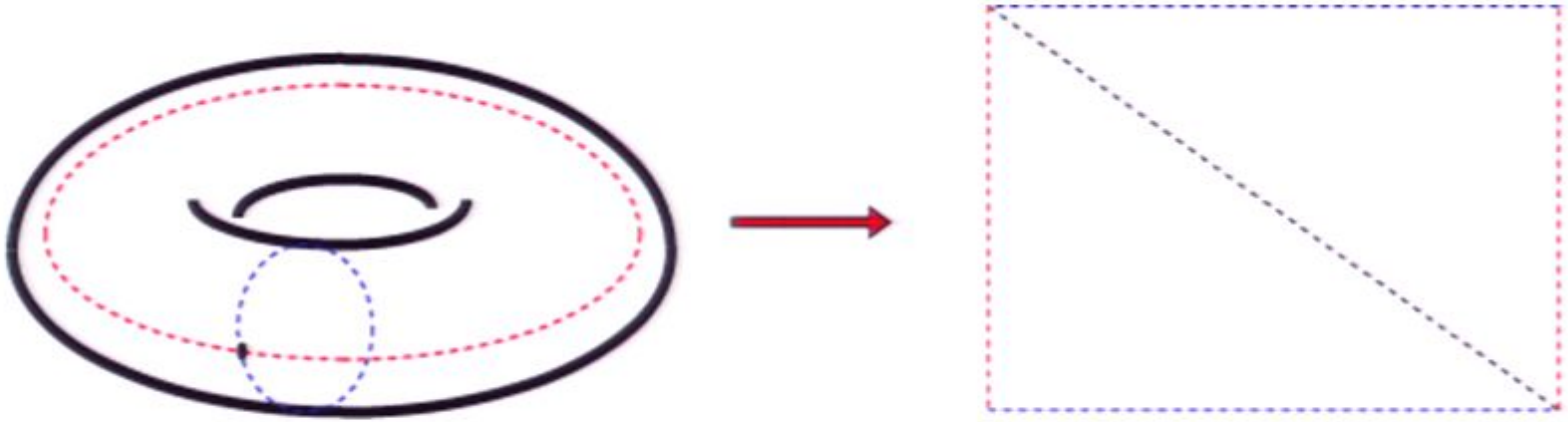
Orientable Surfaces



surface

triangulation

Orientable Surfaces



surface

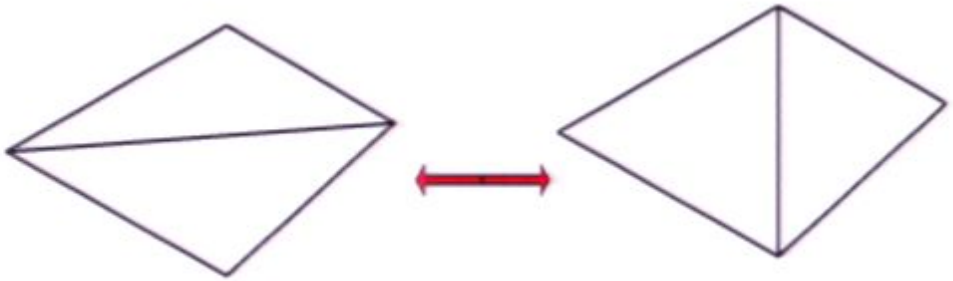
triangulation

Orientable Surfaces

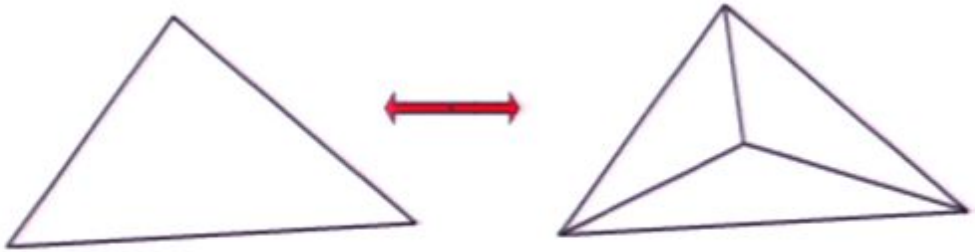
Link diagrams came equipped with “legal moves,” the Reidemeister moves.
What are the “legal moves” on triangulations?

Orientable Surfaces

Link diagrams came equipped with “legal moves,” the Reidemeister moves.
What are the “legal moves” on triangulations? The “Pachner moves”!



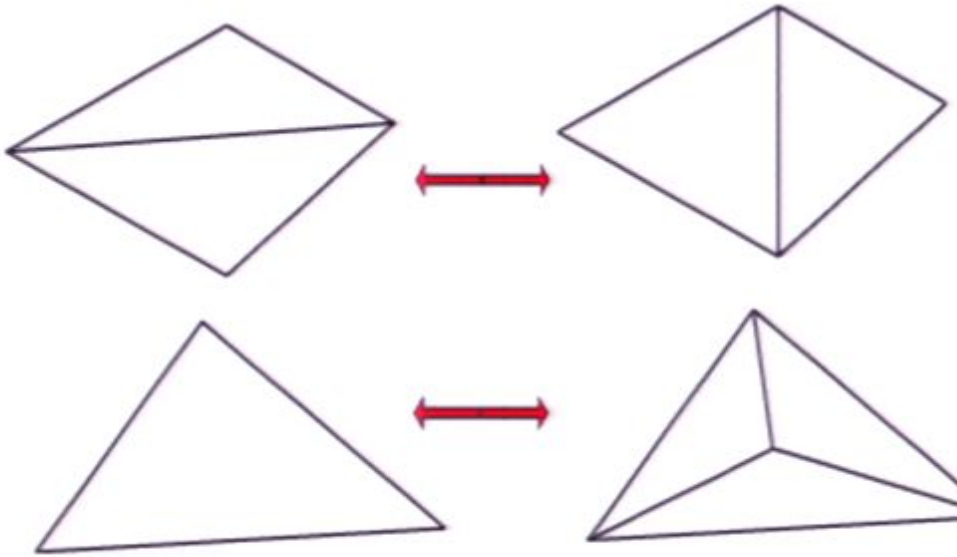
2-2 move



1-3 move

Orientable Surfaces

Link diagrams came equipped with “legal moves,” the Reidemeister moves.
What are the “legal moves” on triangulations? The “Pachner moves”!



2-2 move

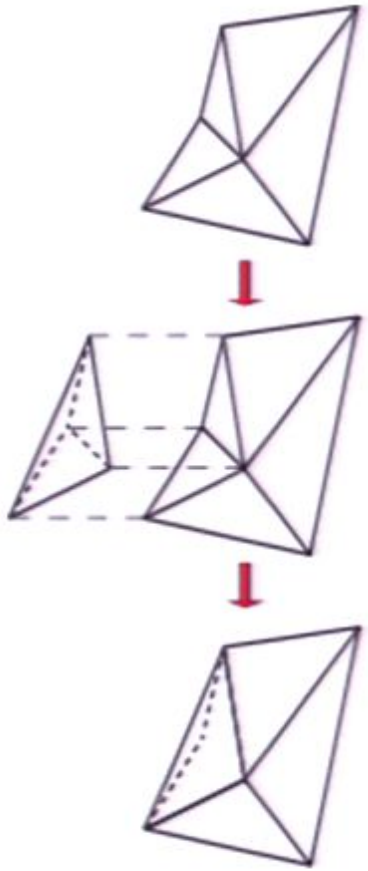
1-3 move

Theorem (Pachner) [1991]:
Any two triangulations of the same surface differ by a finite number of these moves.

Manifolds

What about higher dimensions still? Picture the Pachner moves like this:

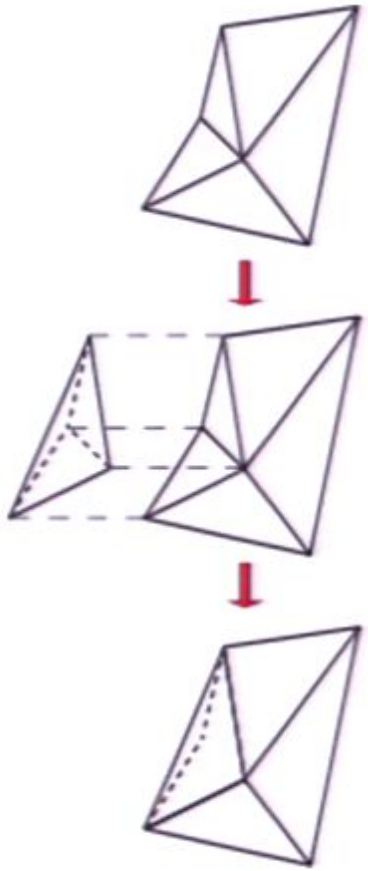
2-2



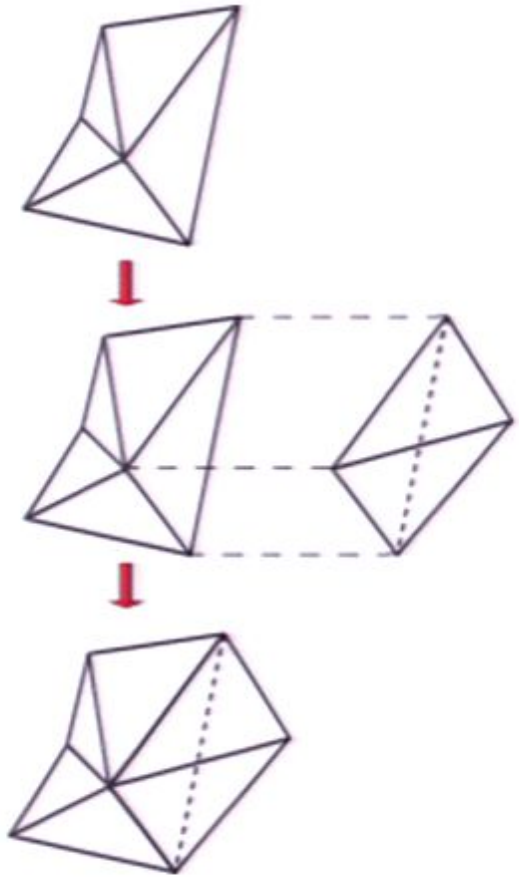
Manifolds

What about higher dimensions still? Picture the Pachner moves like this:

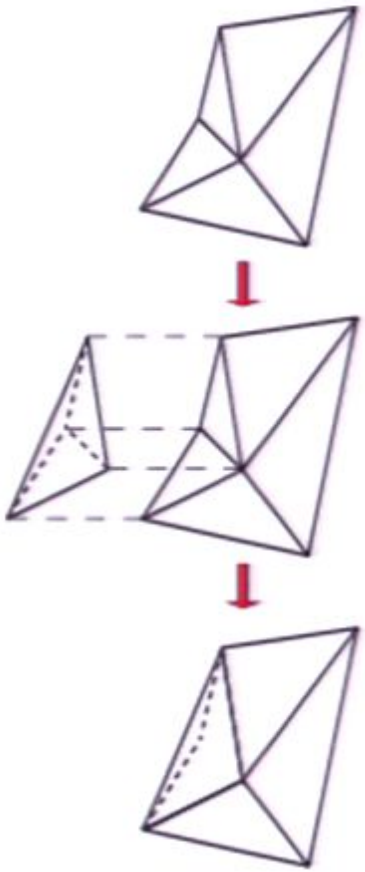
2-2



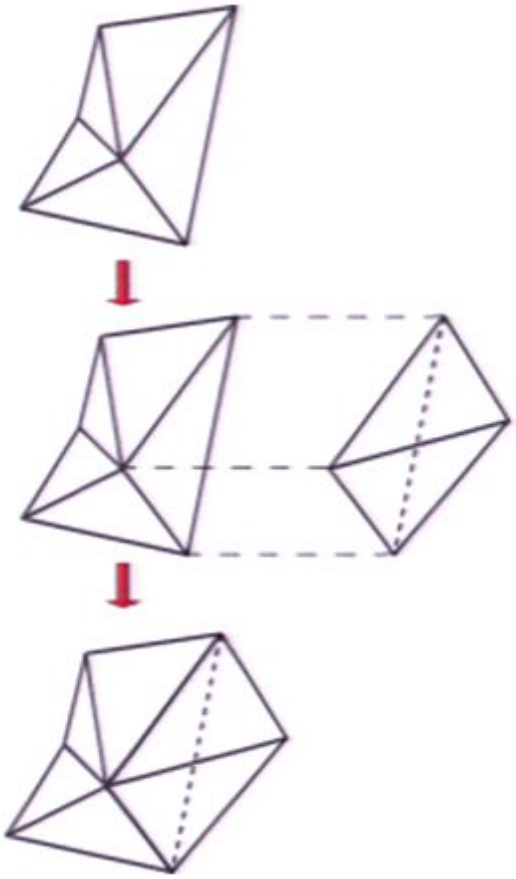
1-3



Manifolds



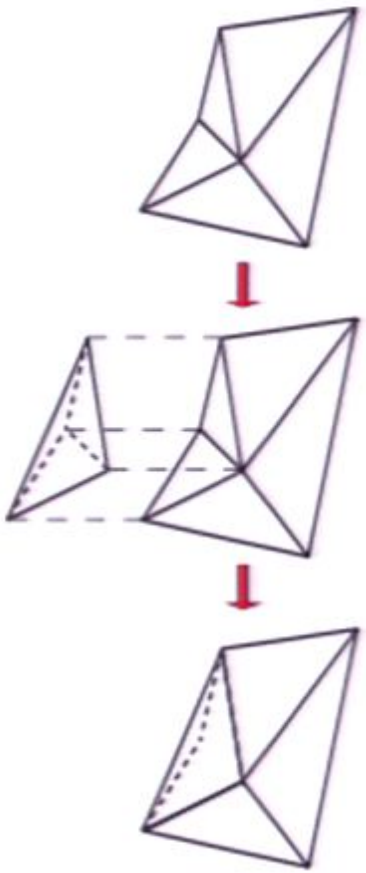
2-2



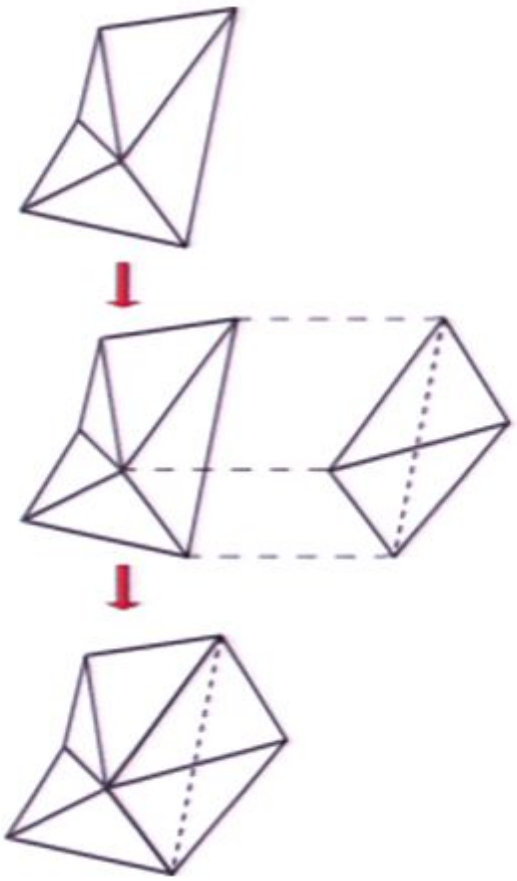
1-3

The n-dimensional Pachner moves are in one-to-one correspondence with the ways of cutting the boundary of an n+1 simplex into two connected pieces.

Manifolds



2-2



1-3

The n-dimensional Pachner moves are in one-to-one correspondence with the ways of cutting the boundary of an n+1 simplex into two connected pieces.

What is this good for?

Ok, so we have a beautiful way to replace **topological** objects with **combinatorial** ones in a natural way.

What is this good for?

What is this good for?

Ok, so we have a beautiful way to replace **topological** objects with **combinatorial** ones in a natural way.

What is this good for?

We're missing **algebra!**

1. replace topology with combinatorics; figure out what the "legal moves" are;

What is this good for?

Ok, so we have a beautiful way to replace **topological** objects with **combinatorial** ones in a natural way.

What is this good for?

We're missing **algebra!**

1. replace topology with combinatorics; figure out what the "legal moves" are;
2. find **tensors** which satisfy these legal moves;
3. decorate the combinatorial structure with the tensors;

What is this good for?

Ok, so we have a beautiful way to replace **topological** objects with **combinatorial** ones in a natural way.

What is this good for?

We're missing **algebra!**

1. replace topology with combinatorics; figure out what the "legal moves" are;
2. find **tensors** which satisfy these legal moves;
3. decorate the combinatorial structure with the tensors;
4. profit!

This is good for a lot of things.

We'll just focus on the fact that it gives us **topological invariants!**

Tensors: definition

Tensors are a convenient way to do multilinear algebra.
Pick a Hilbert space V. Here's an (l, k)-tensor:

$$T \in \underbrace{V \otimes V \otimes \cdots \otimes V}_{l} \otimes \underbrace{V^* \otimes V^* \otimes \cdots \otimes V^*}_{k}$$

T has contravariant rank l and covariant rank k.

Tensors: definition

Tensors are a convenient way to do multilinear algebra.
Pick a Hilbert space V. Here's an (l, k)-tensor:

$$T \in \underbrace{V \otimes V \otimes \cdots \otimes V}_{l} \otimes \underbrace{V^* \otimes V^* \otimes \cdots \otimes V^*}_{k}$$

T has contravariant rank l and covariant rank k.
If we had a basis for V:

$$T = \sum T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; b_{i_1} \otimes b_{i_2} \otimes \cdots \otimes b_{i_l} \otimes b^*_{j_1} \otimes b^*_{j_2} \otimes \cdots b^*_{j_k}$$

… but we will just denote it by $T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k}$.

By following some basic conventions, we can use this notation in a basis-independent way.

Tensors: operations

We can perform two operations on tensors:

1) Tensor product (notice no repeated indices)

$$T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; R^{t_1 t_2 \cdots t_n}_{s_1 s_2 \cdots s_m}$$

This is now a (l+n, k+m)-tensor.

Tensors: definition

Tensors are a convenient way to do multilinear algebra.
Pick a Hilbert space V. Here's an (l, k)-tensor:

$$T \in \underbrace{V \otimes V \otimes \cdots \otimes V}_{l} \otimes \underbrace{V^* \otimes V^* \otimes \cdots \otimes V^*}_{k}$$

T has contravariant rank l and covariant rank k.
If we had a basis for V:

$$T = \sum T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; b_{i_1} \otimes b_{i_2} \otimes \cdots \otimes b_{i_l} \otimes b^*_{j_1} \otimes b^*_{j_2} \otimes \cdots b^*_{j_k}$$

… but we will just denote it by $T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k}$.

By following some basic conventions, we can use this notation in a basis-independent way.

Tensors: operations

We can perform two operations on tensors:

1) Tensor product (notice no repeated indices)

$$T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; R^{t_1 t_2 \cdots t_n}_{s_1 s_2 \cdots s_m}$$

This is now a (l+n, k+m)-tensor.

Tensors: operations

We can perform two operations on tensors:

1) Tensor product (notice no repeated indices)

$$T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; R^{t_1 t_2 \cdots t_n}_{s_1 s_2 \cdots s_m}$$

This is now a (l+n, k+m)-tensor.

2) Contraction (implicit summation over repeated indices)

$$T^{\overbrace{\cdots i \cdots}^{l}}_{\underbrace{\cdots\cdots}_{k}} \; R^{\overbrace{\cdots\cdots}^{n}}_{\underbrace{\cdots i \cdots}_{m}}$$

This is a (l+n-1, k+m-1)-tensor.

Tensors: examples

Here are some basic examples of tensors, in tensor notation:

- A vector: $\mathbf{v}^i$
- A dual vector: $\mathbf{u}_j$
- A linear operator: $\mathbf{A}_i^j$
- Inner product (by contraction): $\mathbf{v}_i\mathbf{u}^i$
- Applying an operator to a vector: $\mathbf{A}_i^j\mathbf{v}^i$
- Matrix product: $\mathbf{A}_i^j\mathbf{B}_j^k$
- Hilbert-Schmidt norm: $\mathbf{A}_i^j\mathbf{A}_j^i$

Tensors: operations

We can perform two operations on tensors:

1) Tensor product (notice no repeated indices)

$$T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; R^{t_1 t_2 \cdots t_n}_{s_1 s_2 \cdots s_m}$$

This is now a (l+n, k+m)-tensor.

2) Contraction (implicit summation over repeated indices)

$$T^{\overbrace{\cdots i \cdots}^{l}}_{\underbrace{\cdots\cdots}_{k}} \; R^{\overbrace{\cdots\cdots}^{n}}_{\underbrace{\cdots i \cdots}_{m}}$$

This is a (l+n-1, k+m-1)-tensor.

Tensors: definition

Tensors are a convenient way to do multilinear algebra.
Pick a Hilbert space V. Here's an (l, k)-tensor:

$$T \in \underbrace{V \otimes V \otimes \cdots \otimes V}_{l} \otimes \underbrace{V^* \otimes V^* \otimes \cdots \otimes V^*}_{k}$$

T has contravariant rank l and covariant rank k.
If we had a basis for V:

$$T = \sum T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k} \; b_{i_1} \otimes b_{i_2} \otimes \cdots \otimes b_{i_l} \otimes b^*_{j_1} \otimes b^*_{j_2} \otimes \cdots b^*_{j_k}$$

… but we will just denote it by $T^{i_1 i_2 \cdots i_l}_{j_1 j_2 \cdots j_k}$.

By following some basic conventions, we can use this notation in a basis-independent way.

Tensors: examples

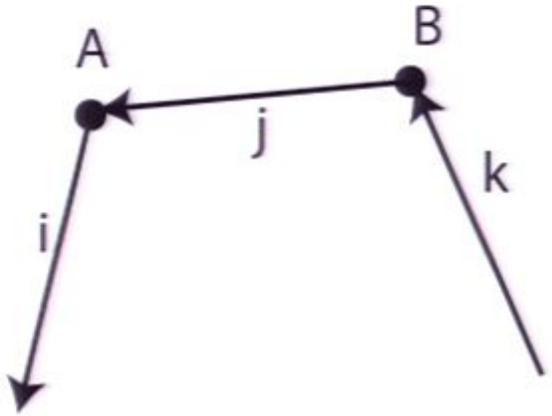
Here are some basic examples of tensors, in tensor notation:

- A vector: $\mathbf{v}^i$
- A dual vector: $\mathbf{u}_j$
- A linear operator: $\mathbf{A}_i^j$
- Inner product (by contraction): $\mathbf{v}_i\mathbf{u}^i$
- Applying an operator to a vector: $\mathbf{A}_i^j\mathbf{v}^i$
- Matrix product: $\mathbf{A}_i^j\mathbf{B}_j^k$
- Hilbert-Schmidt norm: $\mathbf{A}_i^j\mathbf{A}_j^i$

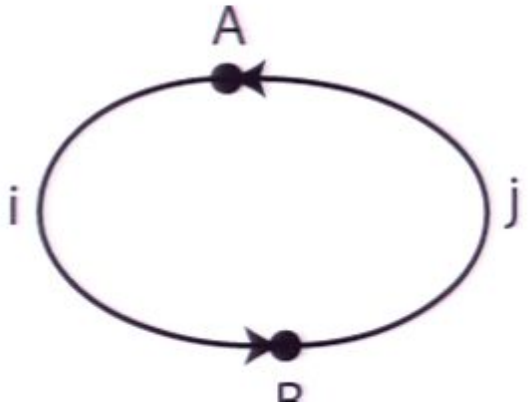
Tensor networks

A **tensor network** is a graph decorated with tensors in the natural way:

- Matrix product: $\mathbf{A}_i^j\mathbf{B}_j^k$



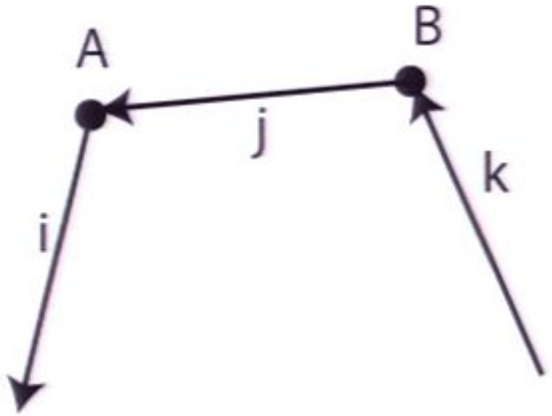
- Trace of a product: $\mathbf{A}_i^j\mathbf{B}_j^i$



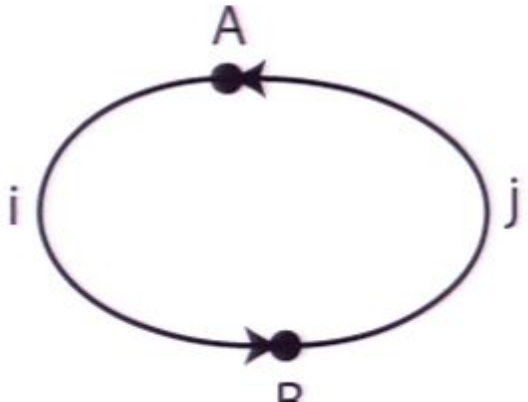
Tensor networks

A **tensor network** is a graph decorated with tensors in the natural way:

- Matrix product: $\mathbf{A}_i^j\mathbf{B}_j^k$



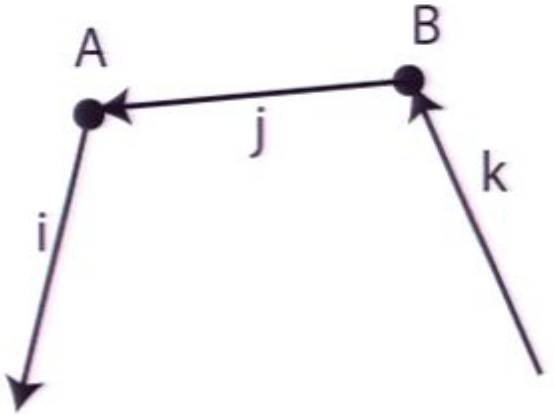
- Trace of a product: $\mathbf{A}_i^j\mathbf{B}_j^i$

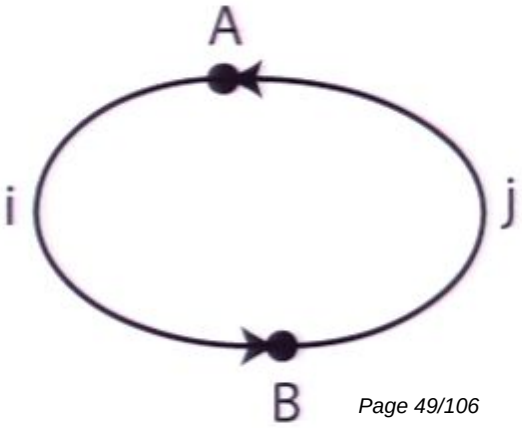


Tensor networks

Two facts about tensor networks:

1) Every network corresponds to a single tensor; rank = (# of incoming edges, # of outgoing edges.)

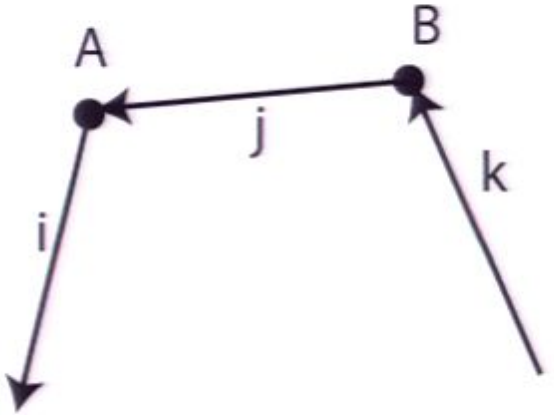




Tensor networks

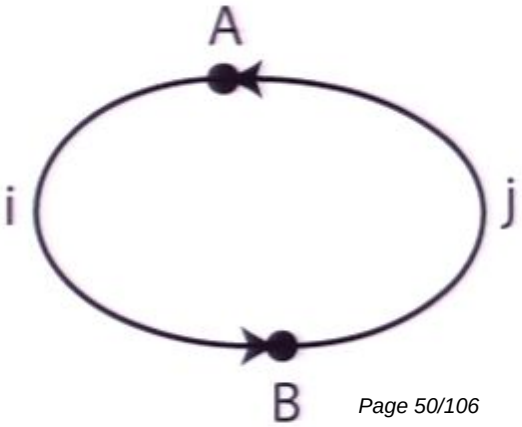
Two facts about tensor networks:

1) Every network corresponds to a single tensor; rank = (# of incoming edges, # of outgoing edges.)



2) A network with no loose edges corresponds to a single complex number:

- ✓ label each edge with a basis vector;
- ✓ each tensor is now a complex number;
- ✓ multiply these numbers together;
- ✓ sum over all possible labelings.

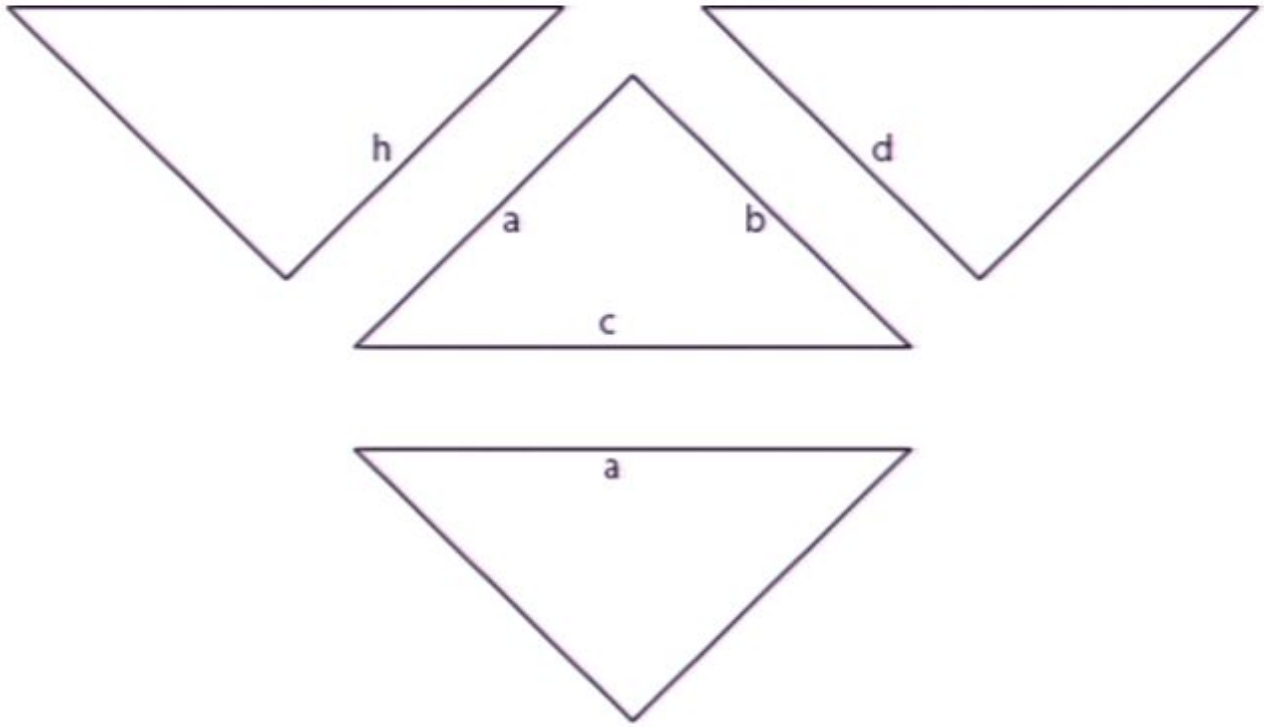


This is called the **contraction** or the **value** of the network.

Decorating triangulations

Decorating a triangulation:

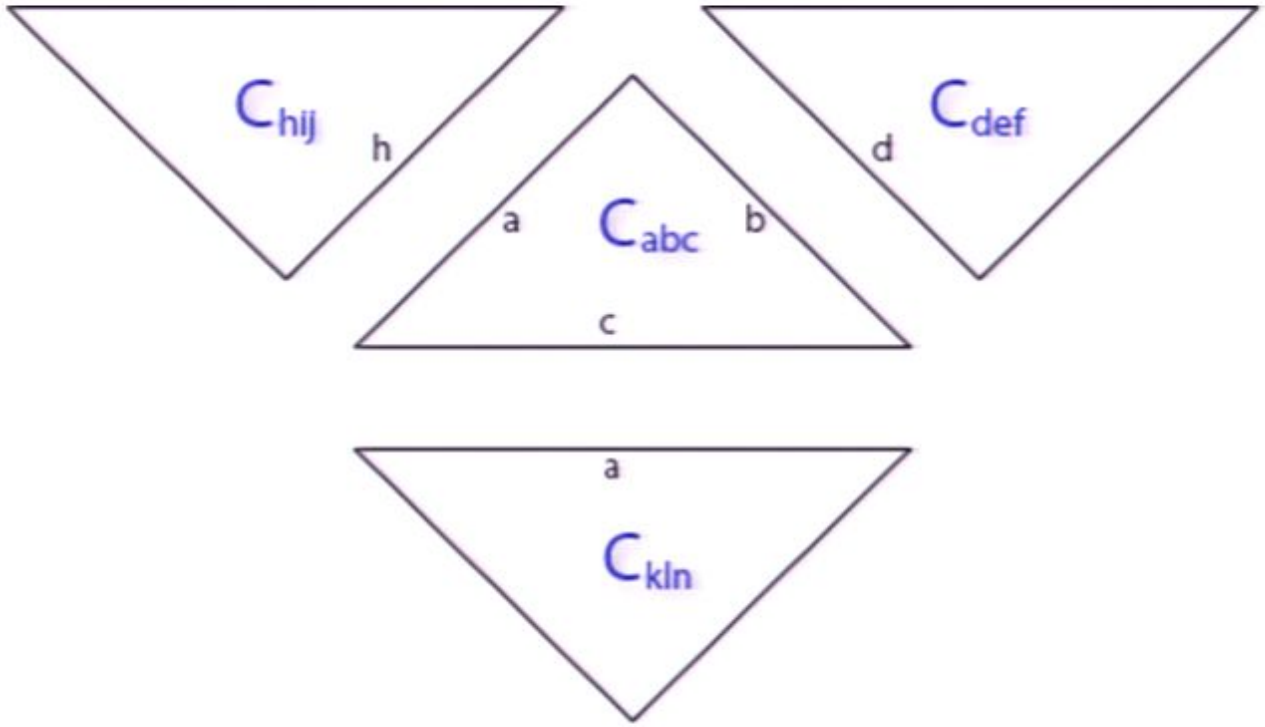
1. label each edge with an index;



Decorating triangulations

Decorating a triangulation:

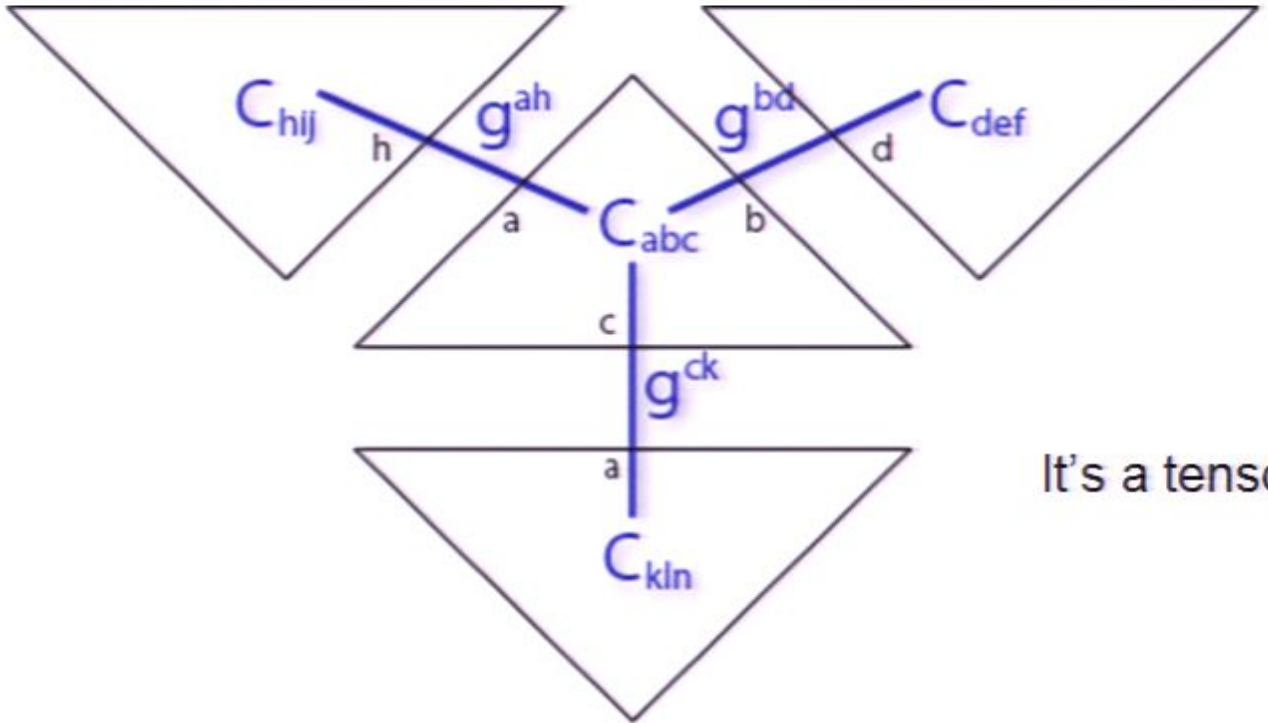
1. label each edge with an index;
2. assign a "spatial" 3-tensor $\mathbf{C}_{ijk}$ to every triangle;



Decorating triangulations

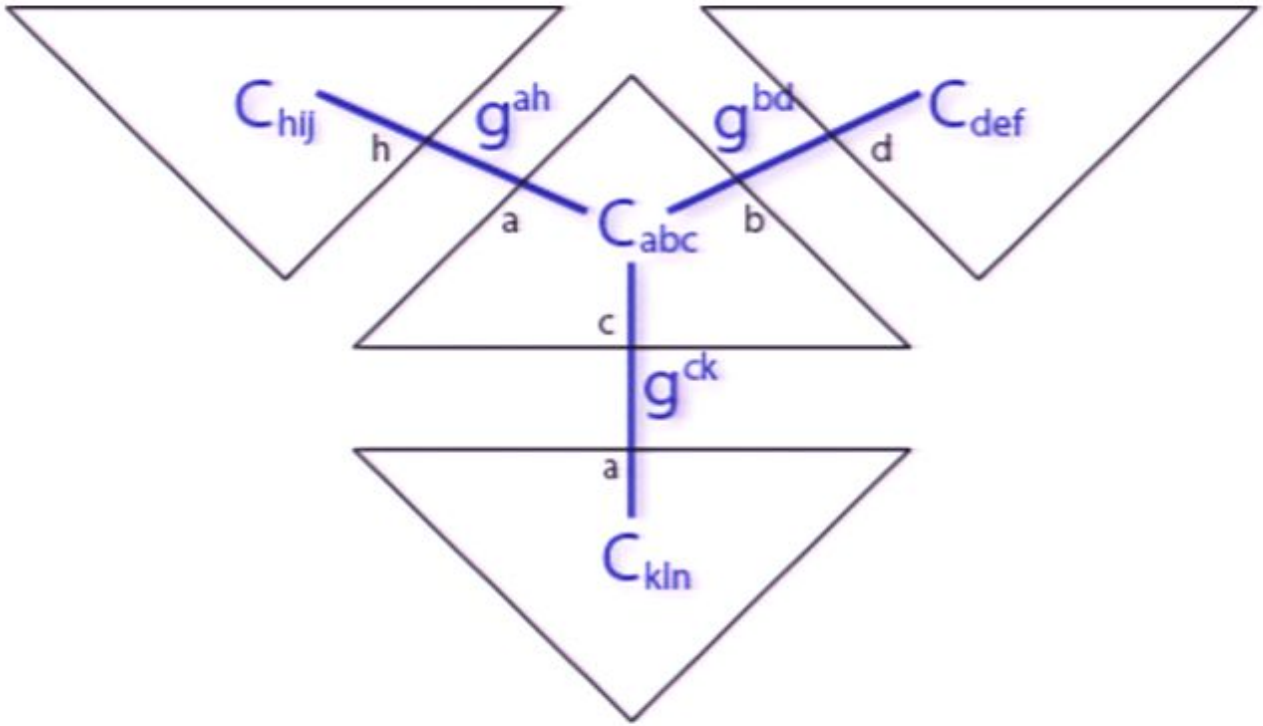
Decorating a triangulation:

1. label each edge with an index;
2. assign a "spatial" 3-tensor $\mathbf{C}_{ijk}$ to every triangle;
3. assign a "gluing" 2-tensor $\mathbf{g}^{ij}$ to every glued pair of edges.



It's a tensor network!

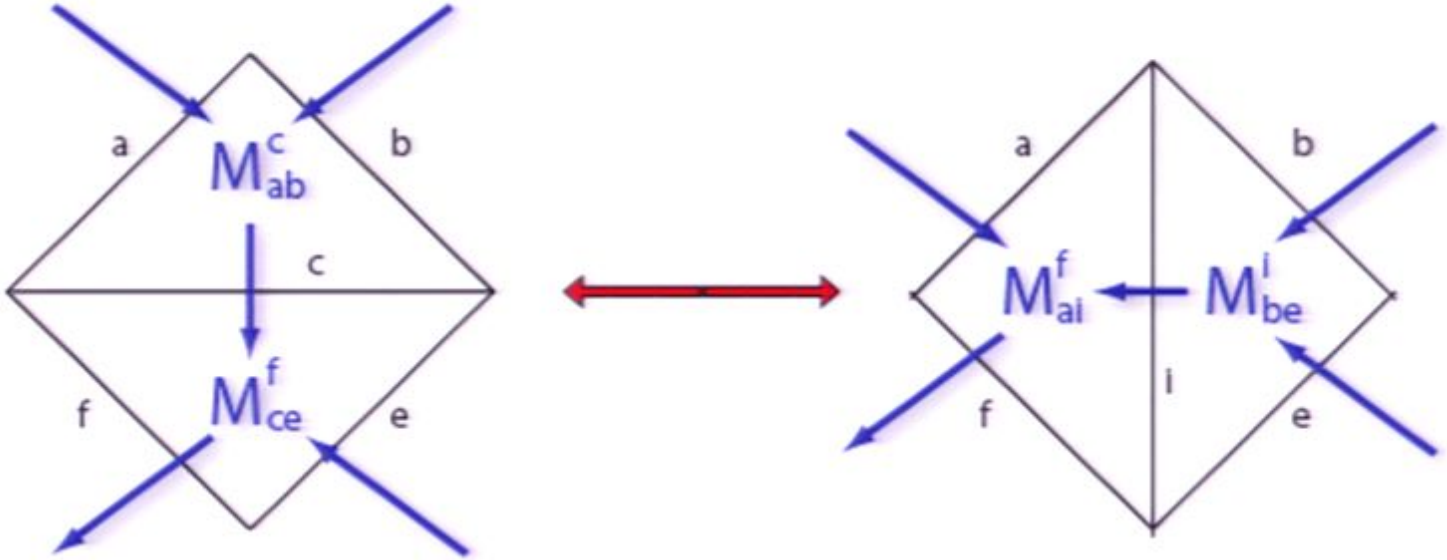
Topological Lattice Field Theories



We want this theory to be topological, so we require that the tensors satisfy the Pachner moves.

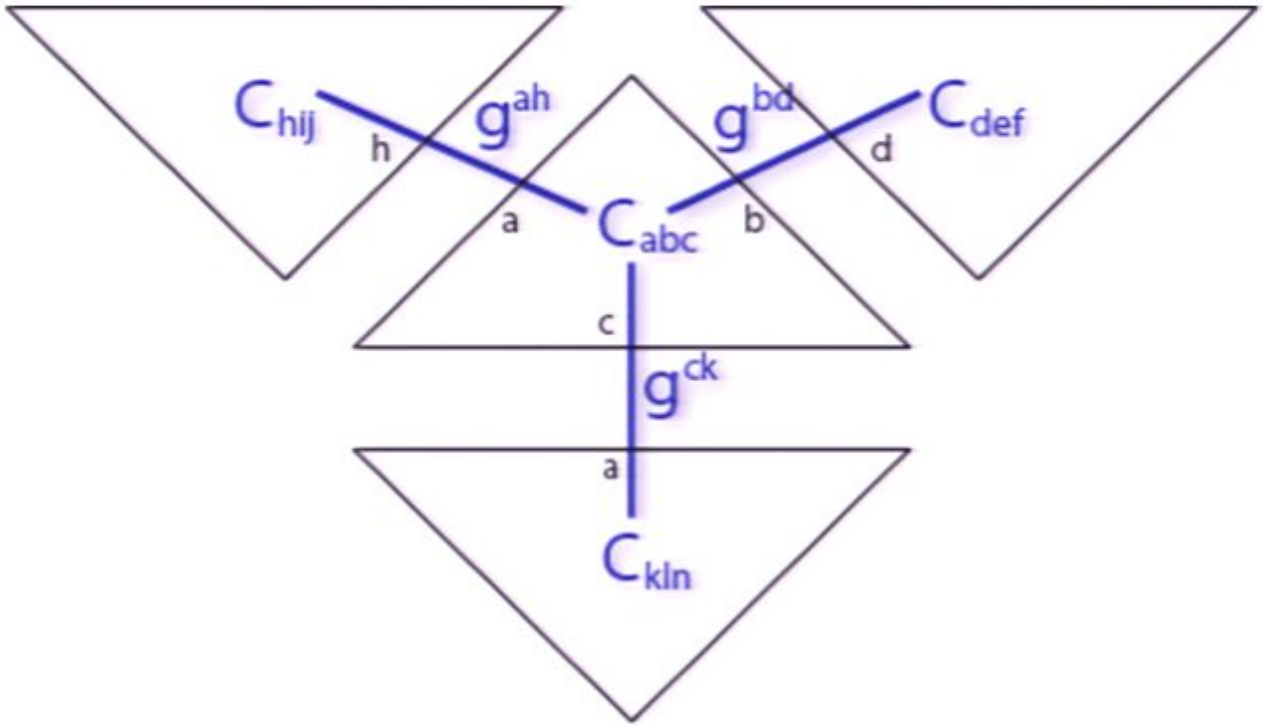
Definition: A 2-D Topological Lattice Field Theory (TLFT) is just a choice of 3-tensor **C** and 2-tensor **g** which satisfy the 2-2 and 1-3 Pachner moves.

Topological Lattice Field Theories



Define $\mathbf{M_{ab}{}^c = C_{abt}g^{tc}}$.

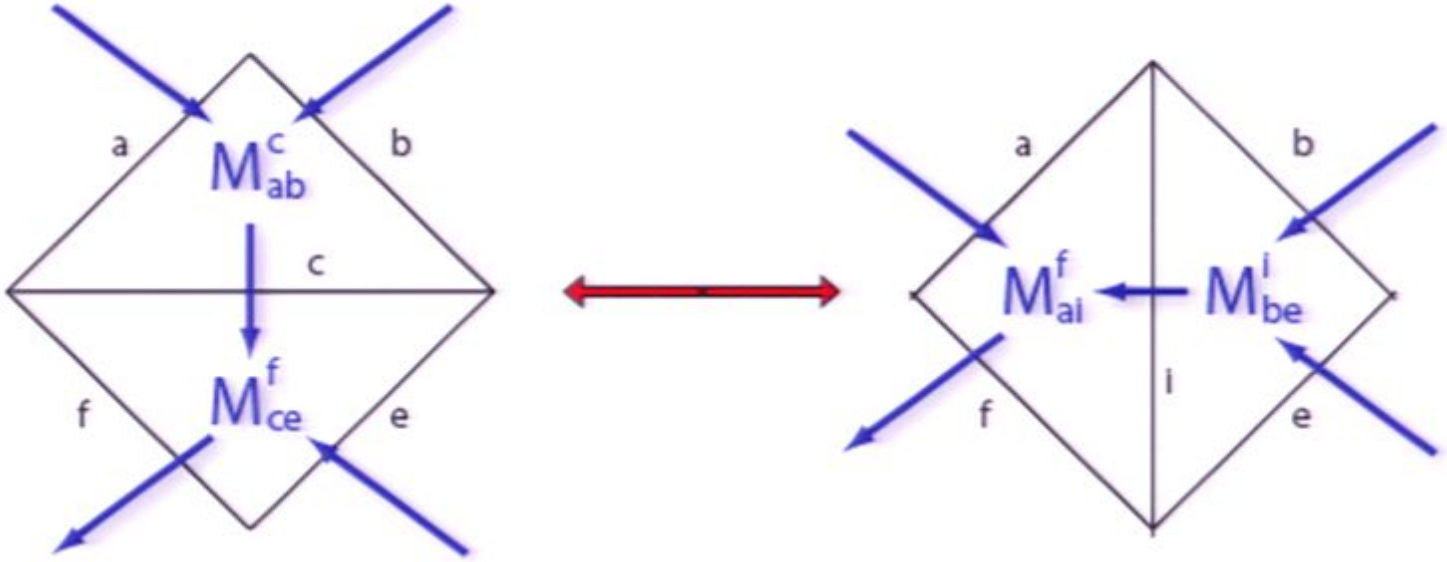
Topological Lattice Field Theories



We want this theory to be topological, so we require that the tensors satisfy the Pachner moves.

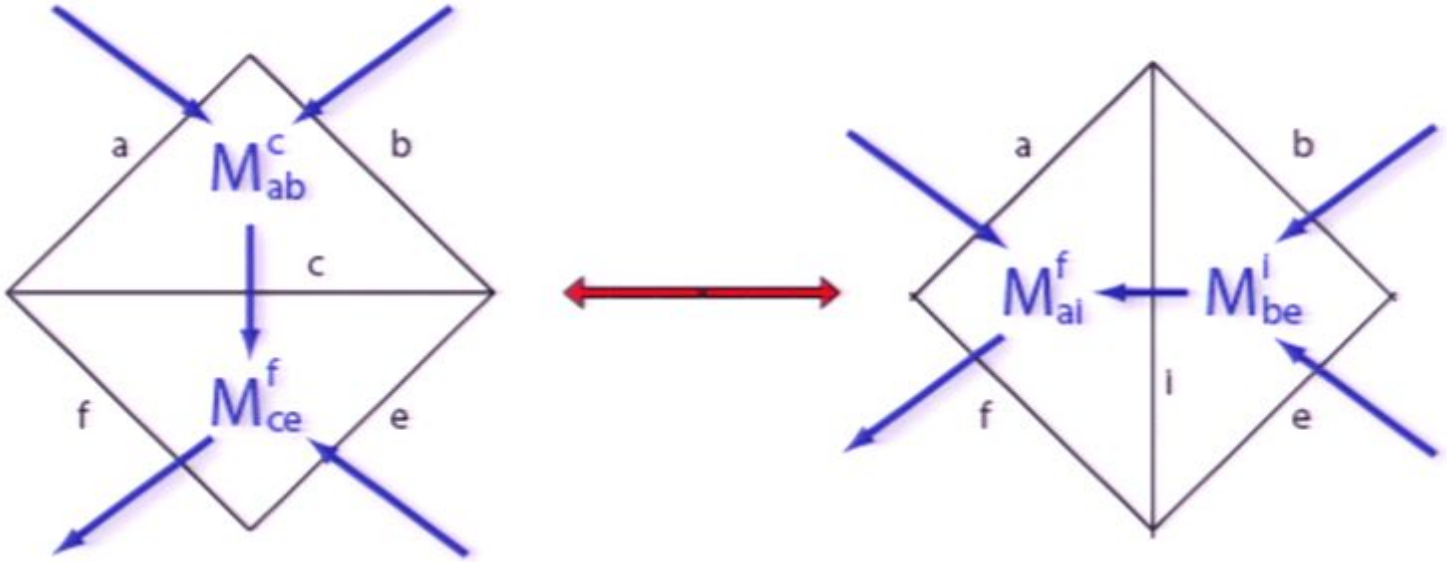
Definition: A 2-D Topological Lattice Field Theory (TLFT) is just a choice of 3-tensor **C** and 2-tensor **g** which satisfy the 2-2 and 1-3 Pachner moves.

Topological Lattice Field Theories



Define $\mathbf{M_{ab}{}^c = C_{abt}g^{tc}}$.

Topological Lattice Field Theories



Define $\mathbf{M_{ab}{}^c = C_{abt}g^{tc}}$. Then the 2-2 move says:

$$\mathbf{M_{ab}{}^c M_{ce}{}^f = M_{be}{}^i M_{ai}{}^f}$$

$$\mathbf{(ab)e = a(be)}$$

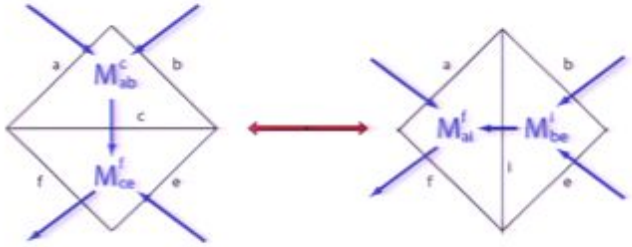
Looks like associativity! What's going on?

Topological Lattice Field Theories

Define $\mathbf{M_{ab}}^{c} = \mathbf{C_{abt}g^{tc}}$. Then the 2-2 move says:

$$\mathbf{M_{ab}}^{c}\mathbf{M_{ce}}^{f} = \mathbf{M_{be}}^{i}\mathbf{M_{ai}}^{f}$$

$$\mathbf{(ab)e = a(be)}$$



Looks like associativity! What's going on?

M makes V into an algebra.

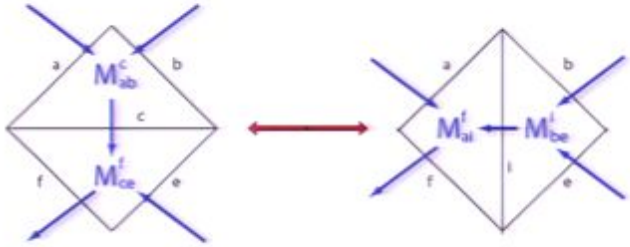
M satisfies 2-2 move iff V is an **associative** algebra!

Topological Lattice Field Theories

Define $\mathbf{M_{ab}}^{c} = \mathbf{C_{abt}}\mathbf{g}^{tc}$. Then the 2-2 move says:

$$\mathbf{M_{ab}}^{c}\mathbf{M_{ce}}^{f} = \mathbf{M_{be}}^{i}\mathbf{M_{ai}}^{f}$$

(ab)e = a(be)



Looks like associativity! What's going on?

M makes V into an algebra.

M satisfies 2-2 move iff V is an **associative** algebra!

One can check that 1-3 move is satisfied iff V is **semisimple**.

Theorem (Fukuma, Hosono, Kawai) [1992]:
2-D Topological Lattice Field Theories are in one-to-one correspondence with semisimple associative algebras.

Group Algebras

What associative, semisimple algebras do we know?

Group algebras!

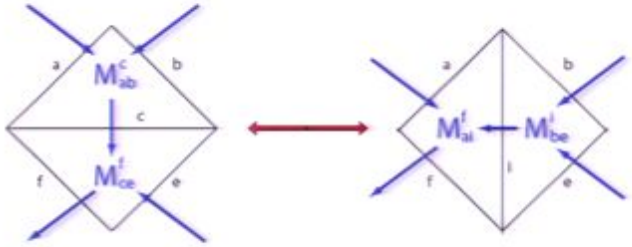
- **G** : finite group
- **CG** : maps from **G** to **C**
- multiplication on **CG** = convolution of functions
- **CG** is associative and semisimple; what are the simple modules?
- They are the **irreducible representations** of **G**.

Topological Lattice Field Theories

Define $\mathbf{M_{ab}}^{c} = \mathbf{C_{abt}}\mathbf{g}^{tc}$. Then the 2-2 move says:

$$\mathbf{M_{ab}}^{c}\mathbf{M_{ce}}^{f} = \mathbf{M_{be}}^{i}\mathbf{M_{ai}}^{f}$$

(ab)e = a(be)



Looks like associativity! What's going on?

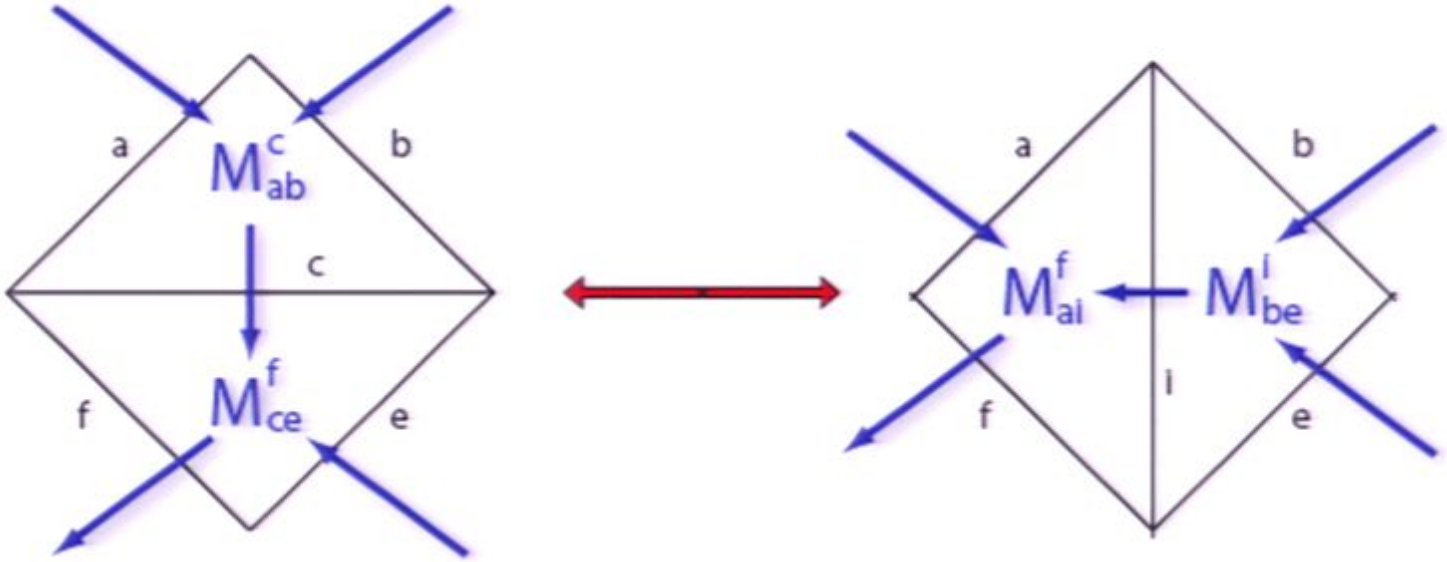
M makes V into an algebra.

M satisfies 2-2 move iff V is an **associative** algebra!

One can check that 1-3 move is satisfied iff V is **semisimple**.

Theorem (Fukuma, Hosono, Kawai) [1992]:
2-D Topological Lattice Field Theories are in one-to-one correspondence with semisimple associative algebras.

Topological Lattice Field Theories



Define $\mathbf{M_{ab}{}^c = C_{abt}g^{tc}}$. Then the 2-2 move says:

$$\mathbf{M_{ab}{}^c M_{ce}{}^f = M_{be}{}^i M_{ai}{}^f}$$

$$\mathbf{(ab)e = a(be)}$$

Looks like associativity! What's going on?

Group Algebras

What associative, semisimple algebras do we know?

Group algebras!

- **G** : finite group
- **CG** : maps from **G** to **C**
- multiplication on **CG** = convolution of functions
- **CG** is associative and semisimple; what are the simple modules?
- They are the **irreducible representations** of **G**.

Group Algebras

What associative, semisimple algebras do we know?

Group algebras!

- **G** : finite group
- **CG** : maps from **G** to **C**
- multiplication on **CG** = convolution of functions
- **CG** is associative and semisimple; what are the simple modules?
- They are the **irreducible representations** of **G**.

Group Algebras

A **representation** of **G** is an action of **G** on a vector space.

Example: symmetric group → permutation matrices.

- span of (1,1,…,1) is fixed;
- this leads to a direct sum decomposition into **irreducible representations (irreps)**.

Group Algebras

A **representation** of **G** is an action of **G** on a vector space.

Example: symmetric group ➔ permutation matrices.

- span of (1,1,…,1) is fixed;
- this leads to a direct sum decomposition into **irreducible representations (irreps)**.

CG is a representation: $\mathbf{x[\ f\ ](y) = f(x^{-1}y)}$.

Every irrep of **G** appears in **CG**, and it appears its dimension many times.

Note: group basis ➔ representation basis map is the **Fourier Transform**.

2d TLFTs with Group Algebras

What happens when we build a TLFT using a group algebra?

3-tensor: $$a \otimes b \otimes c \mapsto \begin{cases} |G| & \text{if } abc = 1 \\ 0 & \text{otherwise.} \end{cases}$$

2-tensor: $$1 \mapsto \frac{1}{|G|} \sum_{g \in G} g \otimes g^{-1}$$

(Alternatively, $\mathbf{M}_{\mathbf{ab}}{}^{\mathbf{c}}$ is just group multiplication!)

2d TLFTs with Group Algebras

What happens when we build a TLFT using a group algebra?

3-tensor: $$a \otimes b \otimes c \mapsto \begin{cases} |G| & \text{if } abc = 1 \\ 0 & \text{otherwise.} \end{cases}$$

2-tensor: $$1 \mapsto \frac{1}{|G|} \sum_{g \in G} g \otimes g^{-1}$$

(Alternatively, $\mathbf{M_{ab}}^{\mathbf{c}}$ is just group multiplication!)

So pick a surface (without boundary) **S** and a group **G**. Take a triangulation of **S**, decorate it with the above tensors of **G**, and contract the whole thing to get a number in **C**...

What in blazes do you get!?

2d TLFTs with Group Algebras

So pick a surface (without boundary) **S** and a group **G**. Take a triangulation of **S**, decorate it with the above tensors of **G**, and contract the whole thing to get a number in **C**… what is this number?

Snyder (2007):

Euler characteristic of **S** = 2-2(genus)

$$\sum_{V \in \hat{G}} \dim(V)^{\chi(S)}$$

irreps of **G**

2d TLFTs with Group Algebras

So pick a surface (without boundary) **S** and a group **G**. Take a triangulation of **S**, decorate it with the above tensors of **G**, and contract the whole thing to get a number in **C**… what is this number?

Snyder (2007):

Euler characteristic of **S** = 2-2(genus)

$$\sum_{V \in \hat{G}} \dim(V)^{\chi(S)} = |G|^{\chi(S)-1} |\mathrm{Hom}(\pi_1(S), G)|$$

irreps of **G**

fundamental group of **S**

2d TLFTs with Group Algebras

So pick a surface (without boundary) **S** and a group **G**. Take a triangulation of **S**, decorate it with the above tensors of **G**, and contract the whole thing to get a number in **C**… what is this number?

Snyder (2007):

Euler characteristic of **S** = 2-2(genus)

$$\sum_{V \in \hat{G}} \dim(V)^{\chi(S)} = |G|^{\chi(S)-1} |\mathrm{Hom}(\pi_1(S), G)|$$

irreps of **G**

fundamental group of **S**

RHS = computed in group basis; LHS = computed in Fourier basis.
This is a famous fact called **Mednykh's equality.** Snyder proved it using TLFTs!

Quantum computation

I promised that all of this would have something to do with quantum computation!

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.

The catch:

- the approximation is additive;
- the approximation scale might be so large that the output is useless;
- it's not always easy to tell how bad the scale is.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.

Implement a linear operator **L** like this:

- Let **UDV** be the singular value decomposition of **L / |L|**;
- We can do **U** and **V**; here's how to do **D**:

$$\begin{pmatrix} D & \sqrt{1-D^2} \\ -\sqrt{1-D^2} & D \end{pmatrix}$$

- Unfortunately, we paid a price when we rescaled by **1 /|L|.**

Quantum computation

I promised that all of this would have something to do with quantum computation!

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.

The catch:

- the approximation is additive;
- the approximation scale might be so large that the output is useless;
- it's not always easy to tell how bad the scale is.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.

Implement a linear operator **L** like this:

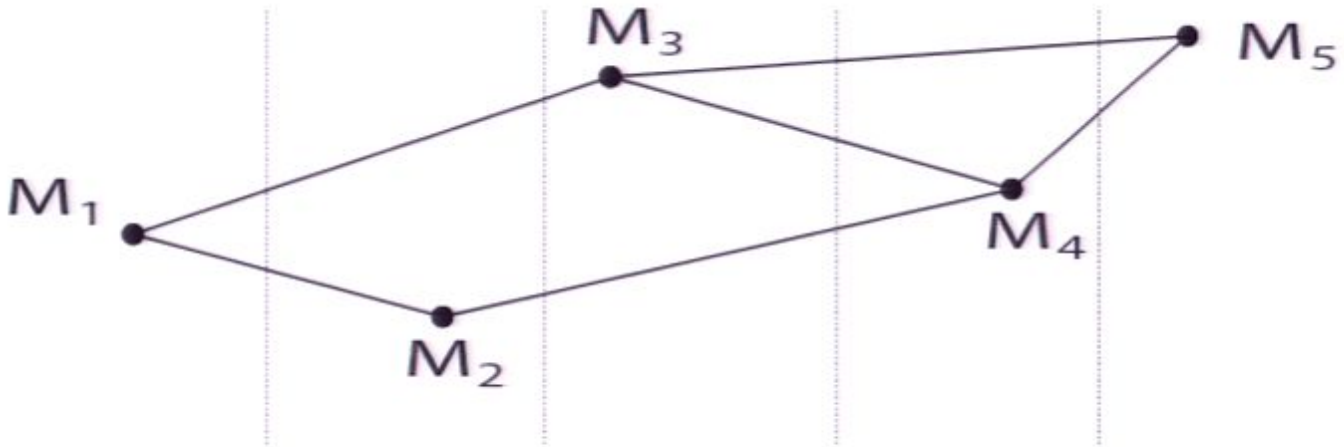
- Let **UDV** be the singular value decomposition of **L** / |**L**|;
- We can do **U** and **V**; here's how to do **D**:

$$\begin{pmatrix} D & \sqrt{1-D^2} \\ -\sqrt{1-D^2} & D \end{pmatrix}$$

- Unfortunately, we paid a price when we rescaled by **1** /|**L**|.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.



- lay the network out like a circuit;
- implement each tensor in sequence as a "zero-padded" linear operator;
- approximation scale is the product of the operator norms of these operators.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.

Implement a linear operator **L** like this:

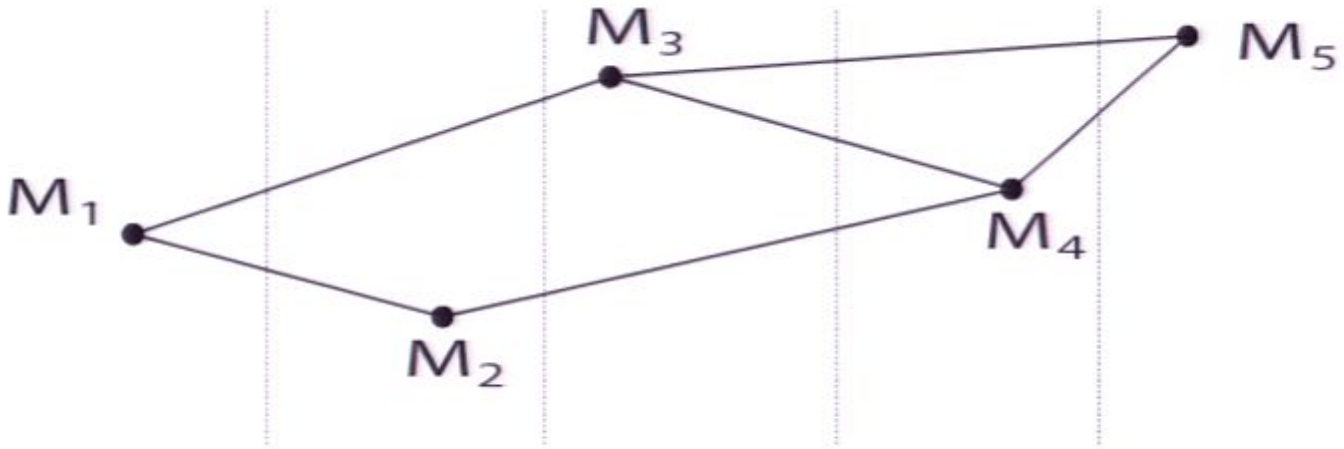
- Let **UDV** be the singular value decomposition of **L / |L|**;
- We can do **U** and **V**; here's how to do **D**:

$$\begin{pmatrix} D & \sqrt{1-D^2} \\ -\sqrt{1-D^2} & D \end{pmatrix}$$

- Unfortunately, we paid a price when we rescaled by **1 /|L|.**

Quantum computation

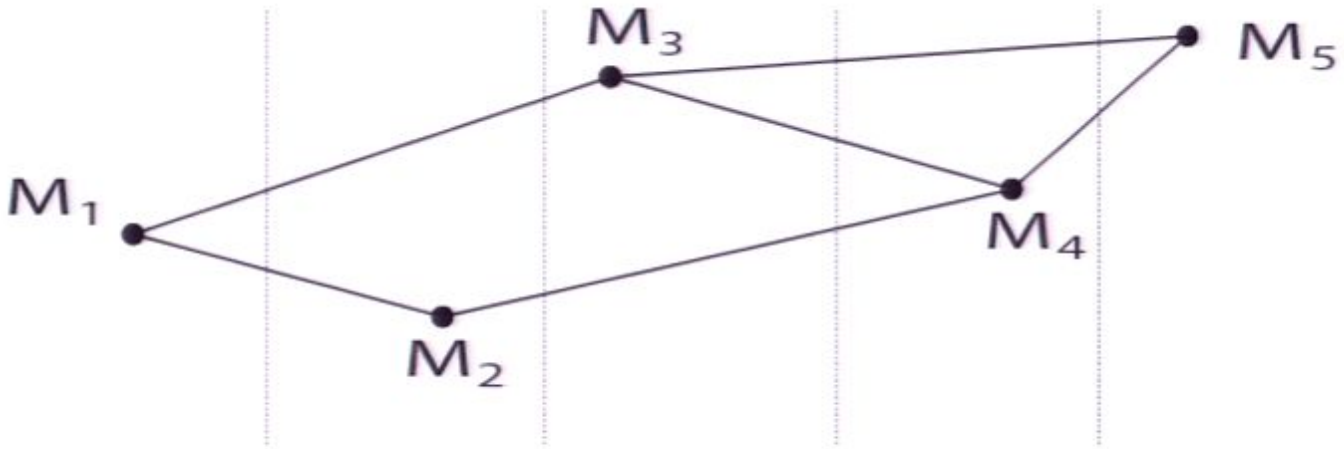
Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.



- lay the network out like a circuit;
- implement each tensor in sequence as a "zero-padded" linear operator;
- approximation scale is the product of the operator norms of these operators.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.



- lay the network out like a circuit;
- implement each tensor in sequence as a "zero-padded" linear operator;
- approximation scale is the product of the operator norms of these operators.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.

Implement a linear operator **L** like this:

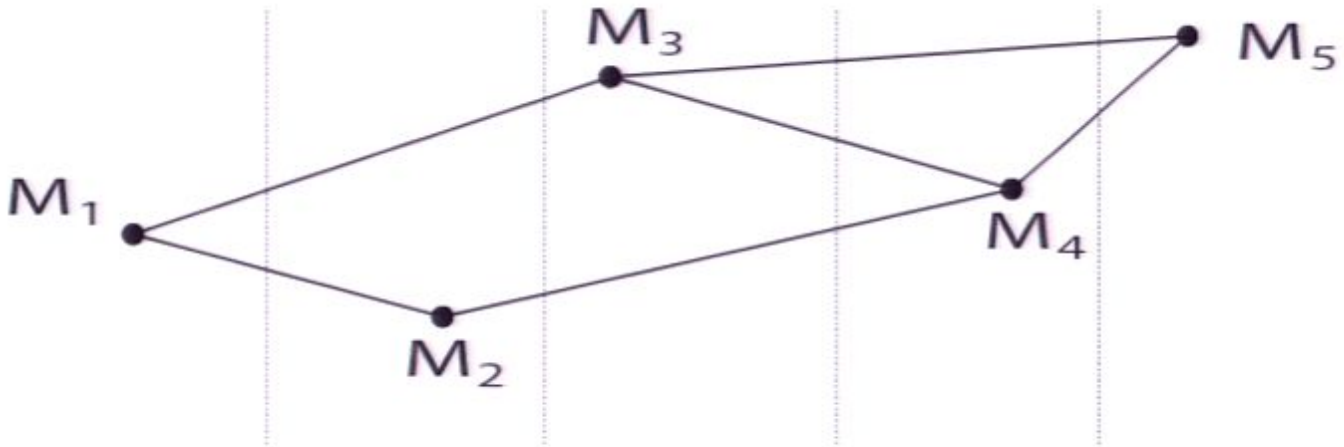
- Let **UDV** be the singular value decomposition of **L** / |**L**|;
- We can do **U** and **V**; here's how to do **D**:

$$\begin{pmatrix} D & \sqrt{1-D^2} \\ -\sqrt{1-D^2} & D \end{pmatrix}$$

- Unfortunately, we paid a price when we rescaled by **1** /|**L**|.

Quantum computation

Theorem (Arad, Landau)[2008]: Quantum computers can approximate the value of tensor networks efficiently.



- lay the network out like a circuit;
- implement each tensor in sequence as a "zero-padded" linear operator;
- approximation scale is the product of the operator norms of these operators.

Quantum algorithm for TLFTs

Input: a triangulation of a surface **S**; a group **G**.

Output: an approximation of $\sum_{V \in \hat{G}} \dim(V)^{\chi(S)} = |G|^{\chi(S)-1} |\mathrm{Hom}(\pi_1(S), G)|$

Algorithm:

1. Put $a \otimes b \otimes c \mapsto \begin{cases} |G| & \text{if } abc = 1 \\ 0 & \text{otherwise.} \end{cases}$ on each face;

2. Put $1 \mapsto \frac{1}{|G|} \sum_{g \in G} g \otimes g^{-1}$ on each pair of glued edges;

3. Contract the resulting network using the Arad-Landau algorithm;

4. Output the result.

Quantum algorithm for TLFTs

The good news:

- can implement these particular tensors in polylog(|G|) time;
- the basic strategy applies to the 3-D case, which is what we really care about!
- the 3-D case involves much harder math!

Quantum algorithm for TLFTs

The good news:

- can implement these particular tensors in polylog(|G|) time;
- the basic strategy applies to the 3-D case, which is what we really care about!
- the 3-D case involves much harder math!

The bad news:

- easy classical algorithm for computing Euler characteristic;
- the speed of the algorithm depends on the quality of the triangulation;
- "obvious" implementation leads to a bad approximation scale;

Quantum algorithm for TLFTs

Input: a triangulation of a surface **S**; a group **G**.

Output: an approximation of $\sum_{V \in \hat{G}} \dim(V)^{\chi(S)} = |G|^{\chi(S)-1} |\mathrm{Hom}(\pi_1(S), G)|$

Algorithm:

1. Put $a \otimes b \otimes c \mapsto \begin{cases} |G| & \text{if } abc = 1 \\ 0 & \text{otherwise.} \end{cases}$ on each face;

2. Put $1 \mapsto \frac{1}{|G|} \sum_{g \in G} g \otimes g^{-1}$ on each pair of glued edges;

3. Contract the resulting network using the Arad-Landau algorithm;

4. Output the result.

Quantum algorithm for TLFTs

The good news:

- can implement these particular tensors in polylog(|G|) time;
- the basic strategy applies to the 3-D case, which is what we really care about!
- the 3-D case involves much harder math!

Quantum algorithm for TLFTs

The good news:

- can implement these particular tensors in polylog(|G|) time;
- the basic strategy applies to the 3-D case, which is what we really care about!
- the 3-D case involves much harder math!

The bad news:

- easy classical algorithm for computing Euler characteristic;
- the speed of the algorithm depends on the quality of the triangulation;
- "obvious" implementation leads to a bad approximation scale;

Quantum algorithm for TLFTs

Input: a triangulation of a surface **S**; a group **G**.

Output: an approximation of $\sum_{V \in \hat{G}} \dim(V)^{\chi(S)} = |G|^{\chi(S)-1} |\mathrm{Hom}(\pi_1(S), G)|$

Algorithm:

1. Put $a \otimes b \otimes c \mapsto \begin{cases} |G| & \text{if } abc = 1 \\ 0 & \text{otherwise.} \end{cases}$ on each face;

2. Put $1 \mapsto \frac{1}{|G|} \sum_{g \in G} g \otimes g^{-1}$ on each pair of glued edges;

3. Contract the resulting network using the Arad-Landau algorithm;

4. Output the result.

Quantum algorithm for TLFTs

The good news:

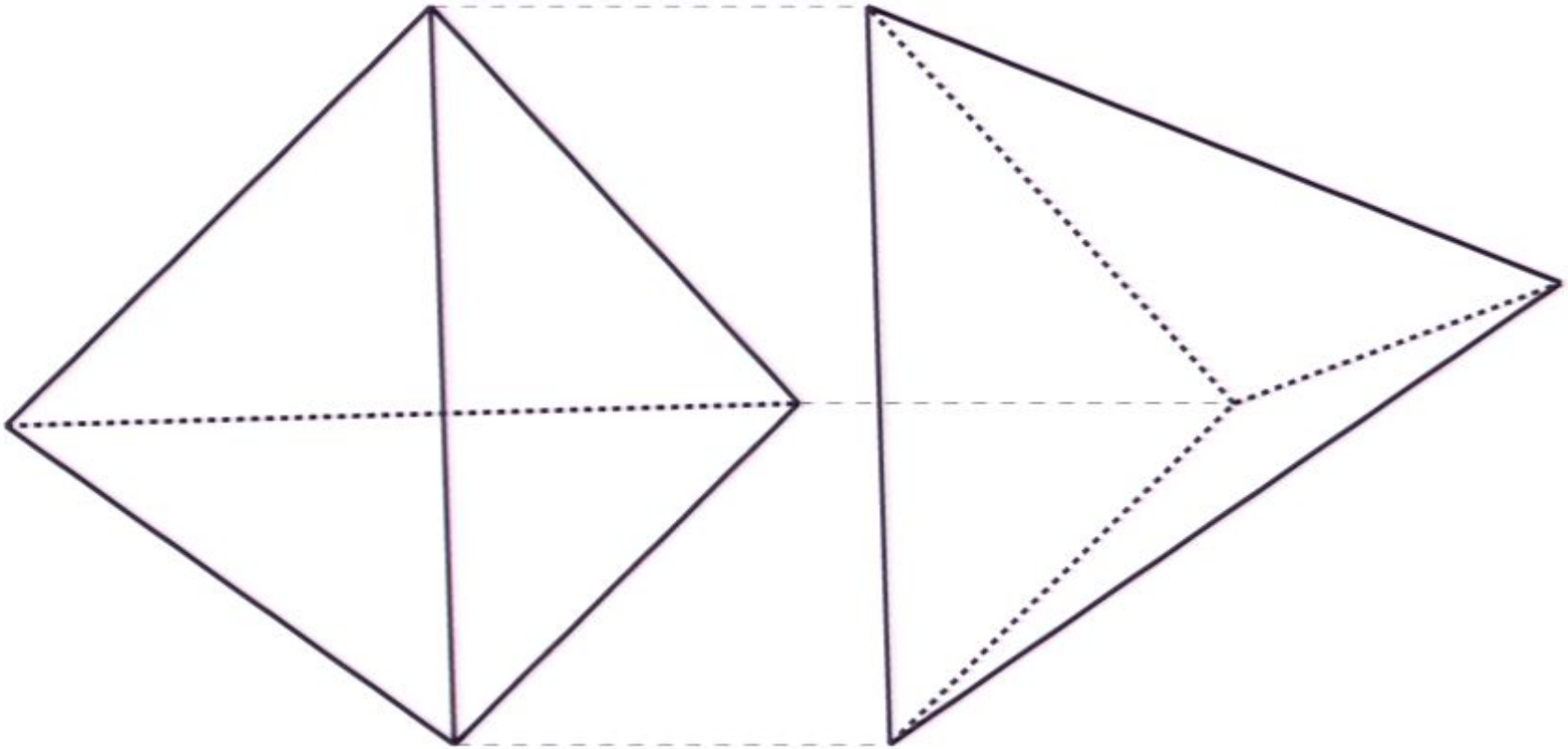
- can implement these particular tensors in polylog(|G|) time;
- the basic strategy applies to the 3-D case, which is what we really care about!
- the 3-D case involves much harder math!

The bad news:

- easy classical algorithm for computing Euler characteristic;
- the speed of the algorithm depends on the quality of the triangulation;
- "obvious" implementation leads to a bad approximation scale;

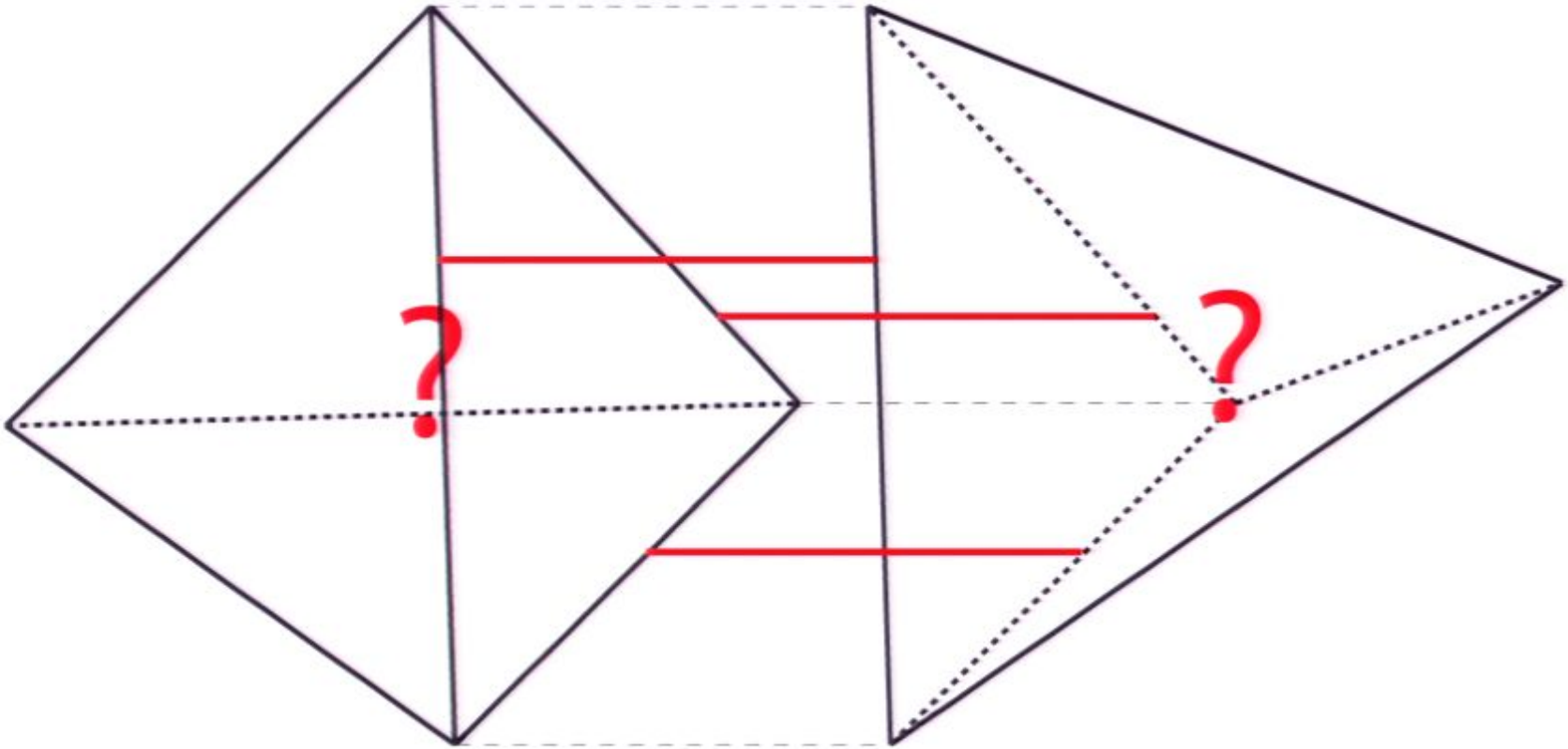
3-D TLFTs

This time, the triangulation will be into tetrahedra...



3-D TLFTs

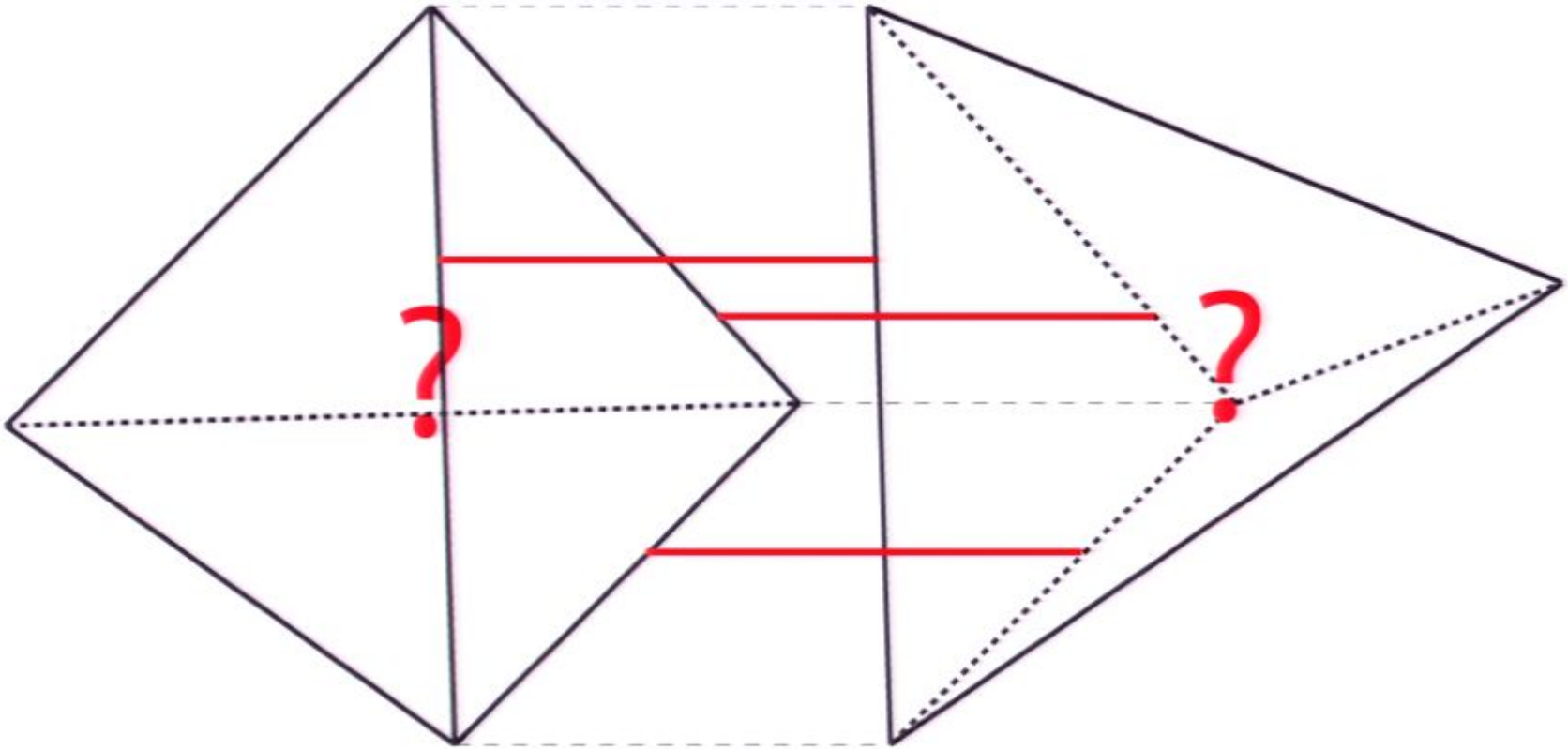
This time, the triangulation will be into tetrahedra...



but what tensors should we use?

3-D TLFTs

This time, the triangulation will be into tetrahedra...



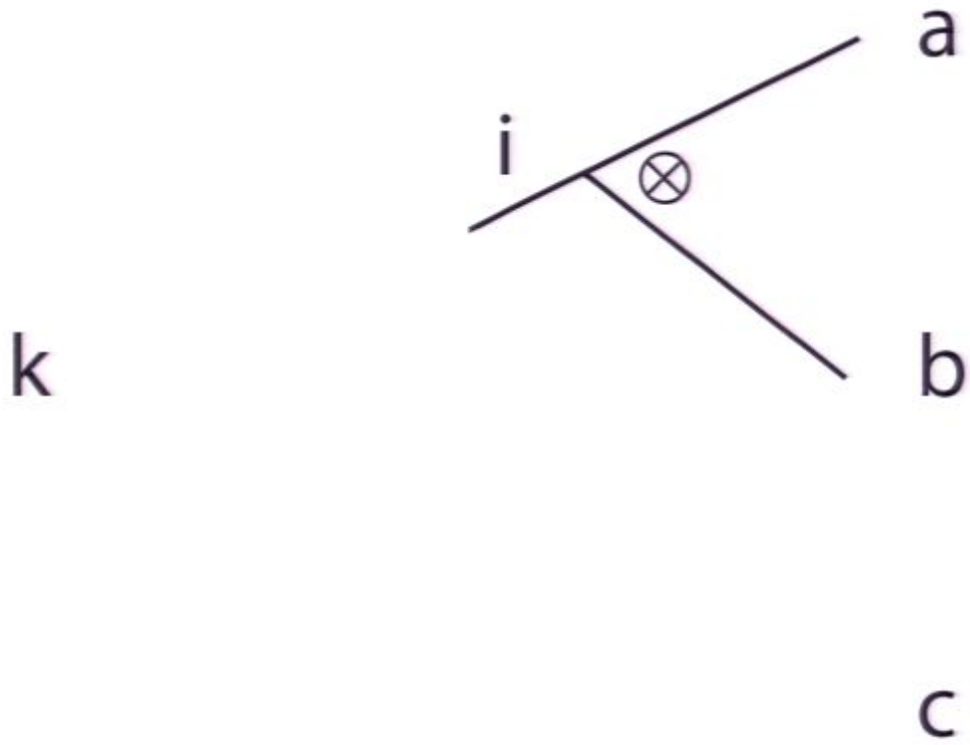
but what tensors should we use?

3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.

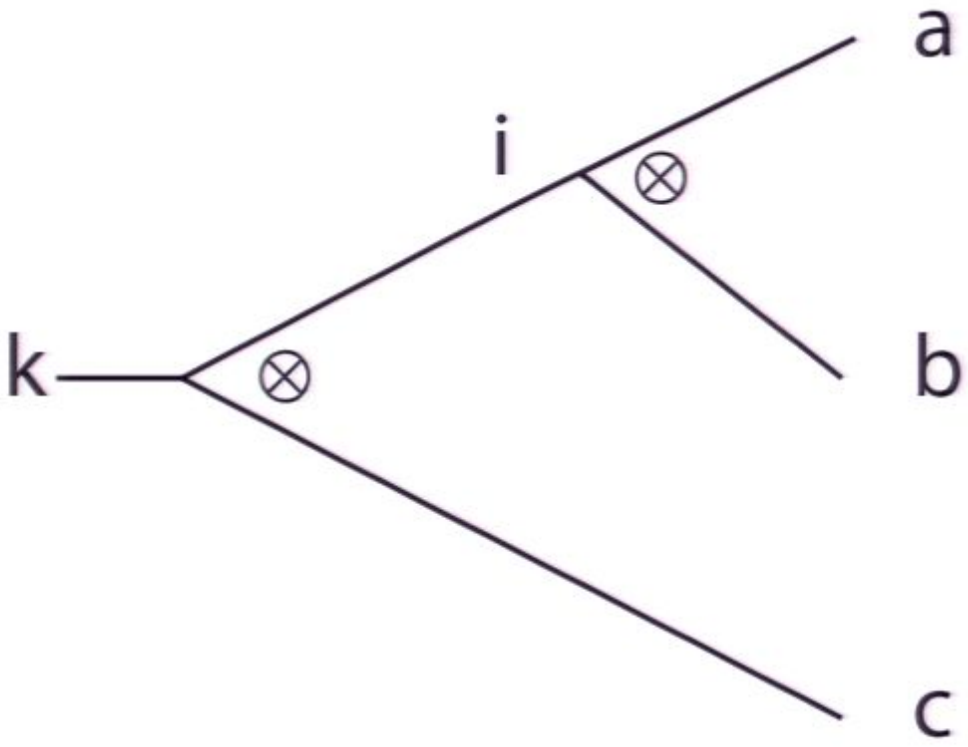
3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.



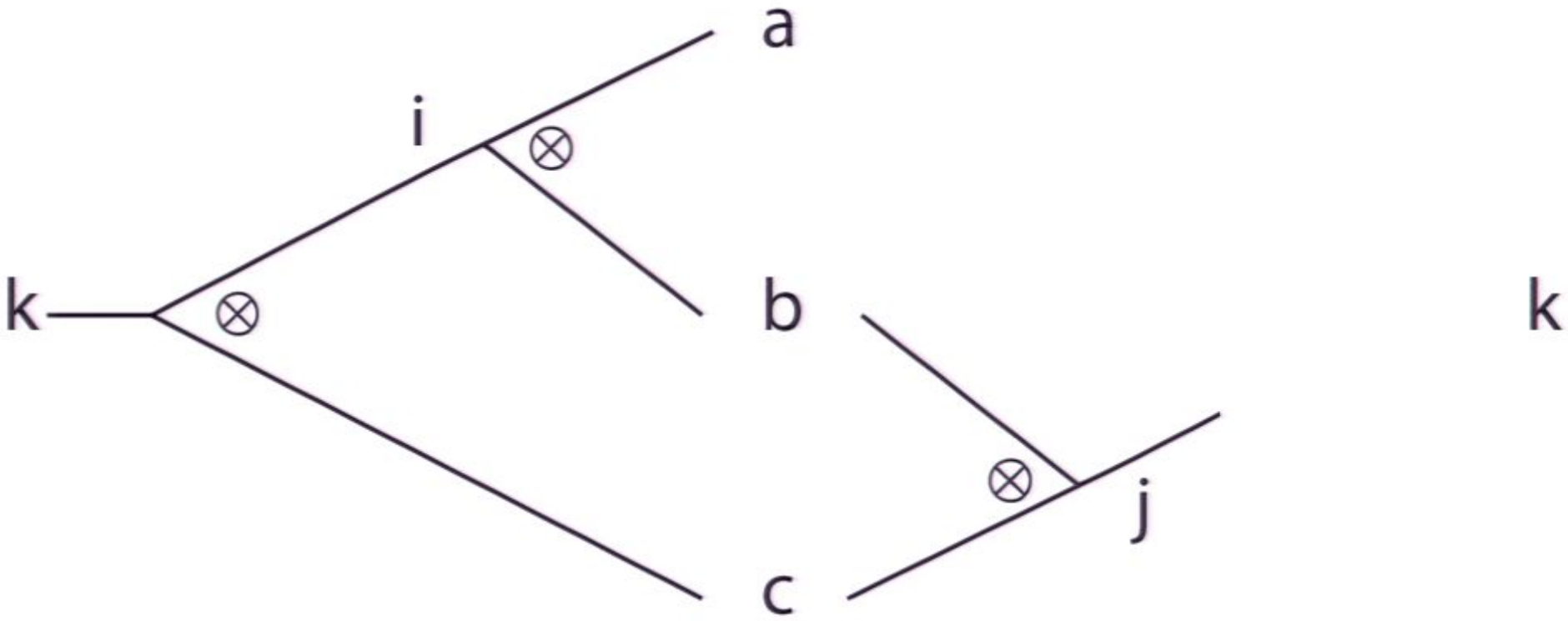
3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.



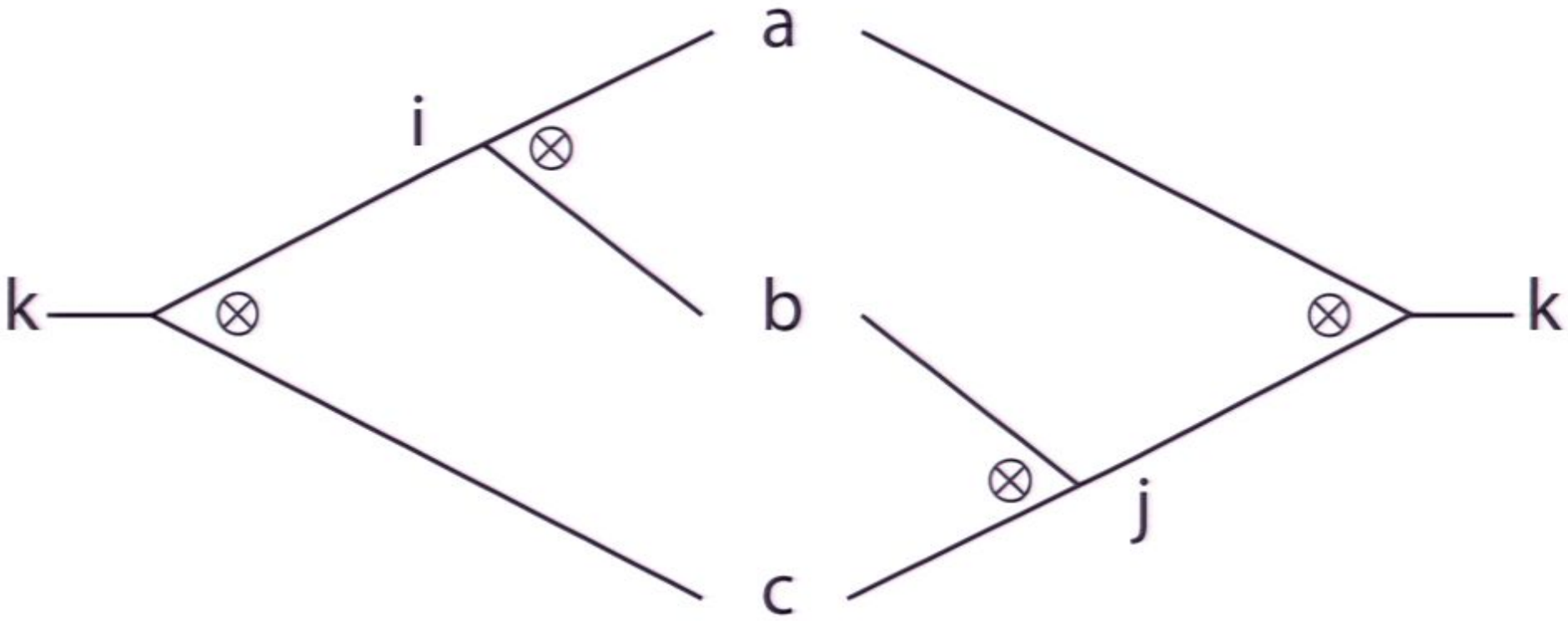
3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.



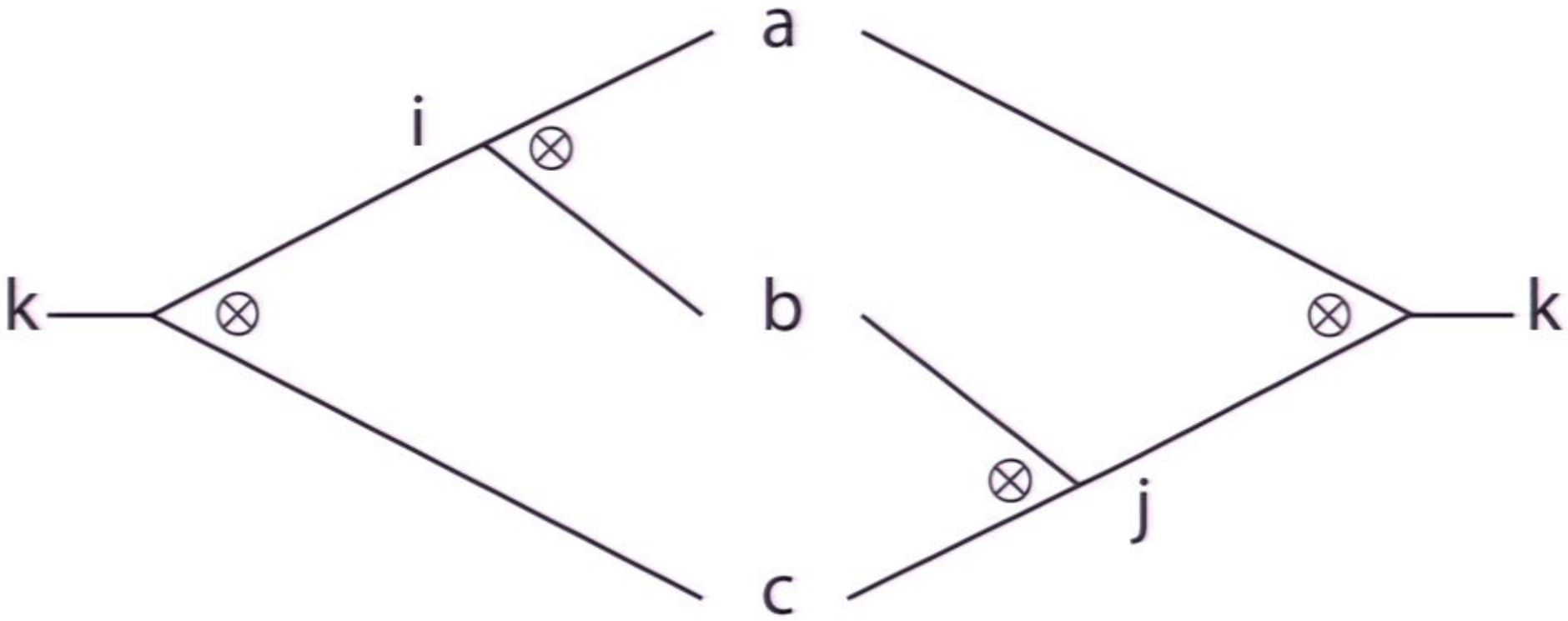
3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.



3-D TLFTs: 6j symbols

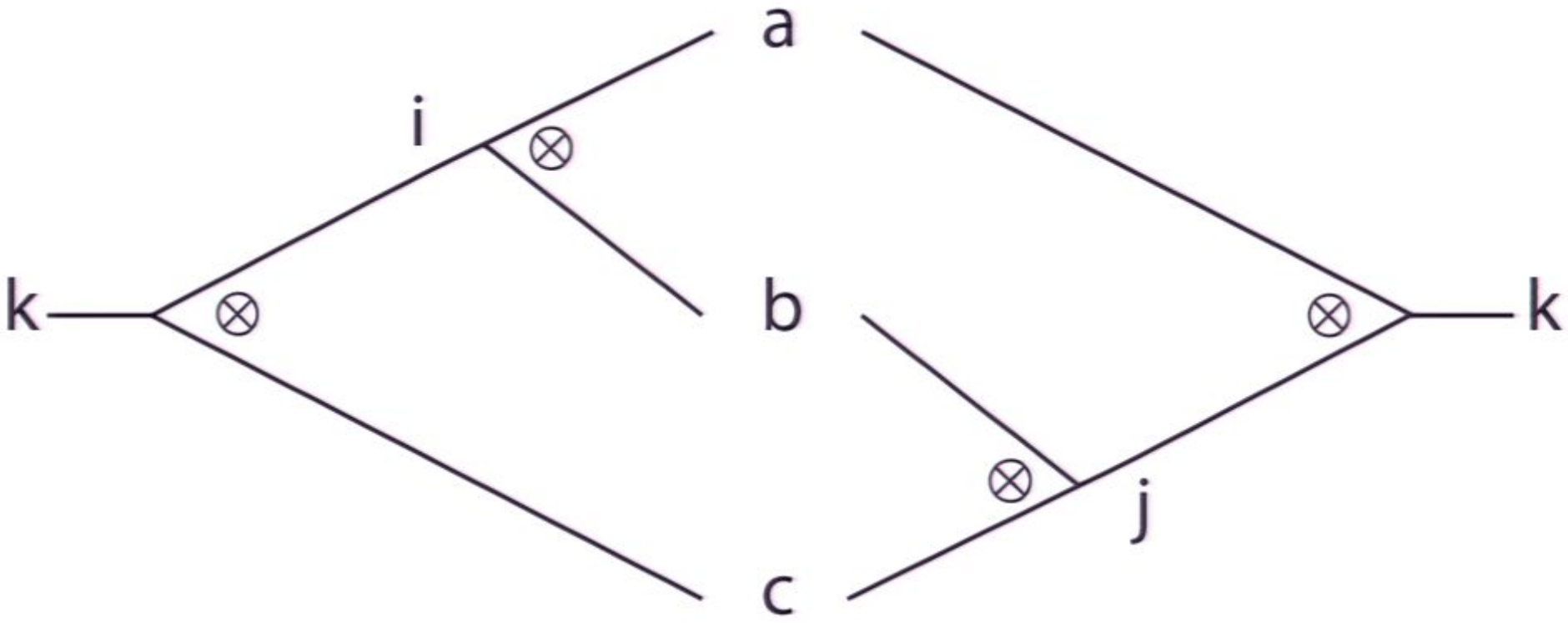
It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.



Schur's Lemma: Any **G**-invariant map from an irrep to itself is scalar multiplication.

3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.



Biedenharn-Elliott identity: These 6j symbols satisfy the 3-D Pachner moves.

3-D TLFTs: 6j symbols

So now we…

- start with a tetrahedral triangulation of a 3-manifold;
- label the edges of all the tetrahedra with **SU(2)** representation names (spins);
- assign a 6j symbol to every tetrahedron;

3-D TLFTs: 6j symbols

So now we…

- start with a tetrahedral triangulation of a 3-manifold;
- label the edges of all the tetrahedra with **SU(2)** representation names (spins);
- assign a 6j symbol to every tetrahedron;

… but now what? There's some problems:

- Many tetrahedra incident on one edge ➔ "hinge" tensor.
- **SU(2)** has infinitely many irreps, so we need $\mathbf{SU_q(2)}$…

3-D TLFTs: 6j symbols

So now we…

- start with a tetrahedral triangulation of a 3-manifold;
- label the edges of all the tetrahedra with **SU(2)** representation names (spins);
- assign a 6j symbol to every tetrahedron;

… but now what? There's some problems:

- Many tetrahedra incident on one edge ➔ "hinge" tensor.
- **SU(2)** has infinitely many irreps, so we need $\mathbf{SU_q(2)}$…

So the situation is more complicated than 2-D, but it can be worked out…

The result is the Turaev-Viro 3-manifold invariant.

3-D TLFTs: 6j symbols

Open: Is there an efficient quantum algorithm for the Turaev-Viro invariant?

THANKS!

Quantum algorithm for TLFTs

Input: a triangulation of a surface **S**; a group **G**.

Output: an approximation of $\sum_{V \in \hat{G}} \dim(V)^{\chi(S)} = |G|^{\chi(S)-1} |\mathrm{Hom}(\pi_1(S), G)|$

Algorithm:

1. Put $a \otimes b \otimes c \mapsto \begin{cases} |G| & \text{if } abc = 1 \\ 0 & \text{otherwise.} \end{cases}$ on each face;

2. Put $1 \mapsto \frac{1}{|G|} \sum_{g \in G} g \otimes g^{-1}$ on each pair of glued edges;

3. Contract the resulting network using the Arad-Landau algorithm;

4. Output the result.

3-D TLFTs: 6j symbols

It turns out that one should glue along edges; the right 6-tensor comes from the representation theory of **SU(2)** and its quantum variants.