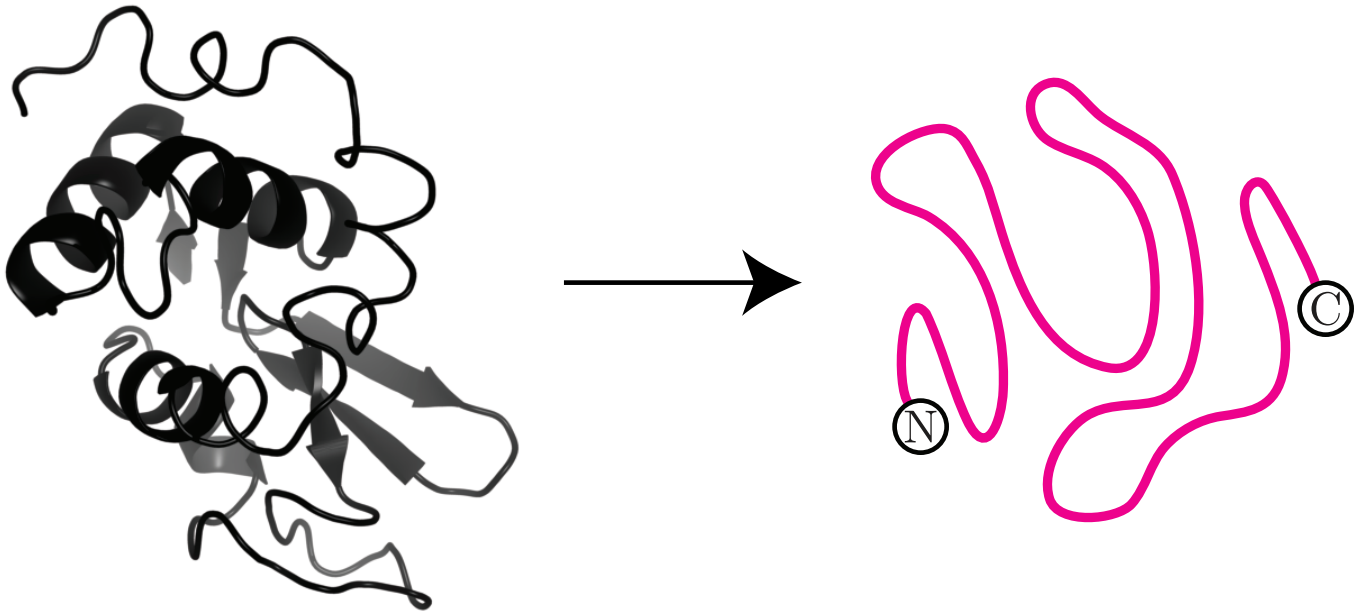
A tile model of entangled proteins

Erica Flapan, Helen Wong, Alireza Mashaghi

September 30, 2024

MIF++ Seminar
Virtually at University of Liverpool

Until the mid-1990's, all native protein structures were assumed to be topologically simple.

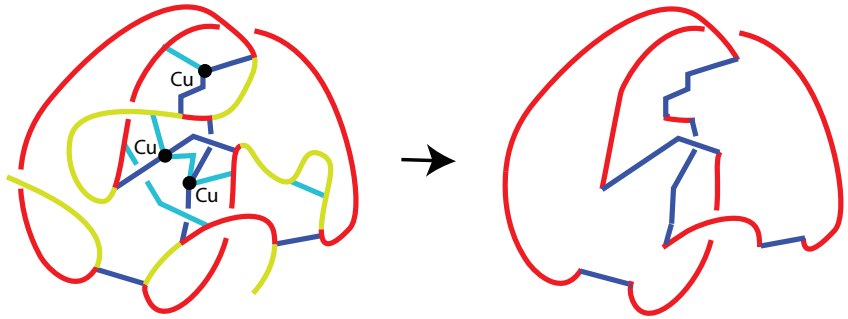


In 1994, the biochemist Marc Mansfield postulated that knotted proteins could not exist.

A Protein Knot

In 1995, Liang and Mislow identified the first knotted protein by including disulfide bonds and copper atoms in addition to the backbone.

Ascorbate acidase
from zucchini



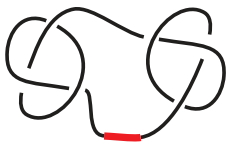
Currently, there are over 200,000 proteins with experimentally solved 3D structures and 1,612 of them have knotted backbones.

Using machine learning techniques, the recent databases AlphaFold and ESMFold predicted 3D structures for hundreds of millions of proteins.

This includes 700K proteins whose backbone contains complex knots that had not been previously predicted.

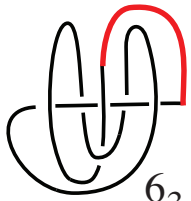
Knotted backbones

Some knotted protein chains (without red arcs) that have been solved or predicted:

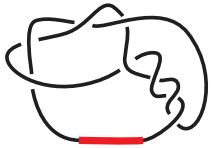


composite knot $3_1 \# 3_1$

twist knot 6_1



6_3 knot



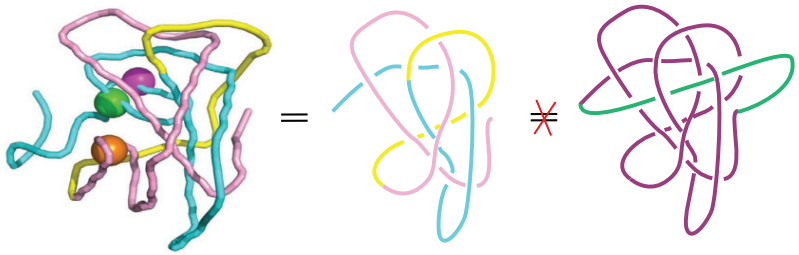
8_3 knot

Most knotted proteins aren't circular, but the energy required to unthread them is large.

To identify the type of knot, we need to join the ends to create a closed loop.

We extend the ends to ∞ in hundreds of directions and choose the knot type occurring most frequently.

For example, we don't choose to connect the ends like this:



Do protein knots serve a purpose?

It is not known whether knots serve a purpose or are just a random occurrence that's been preserved by evolution.

Analogously, it is not known whether human earlobes serve a purpose or are just a random occurrence that's been preserved by evolution.

If knots were a random occurrence, we wouldn't expect the same protein to be knotted in distinct organisms.

However:

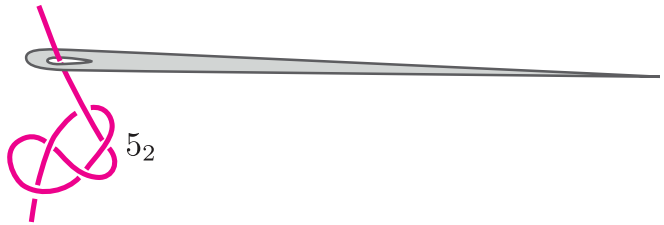
- Carbonic anhydrase containing the 3_1 knot is found in humans, bacteria, and algae.
- Class II ketol-acid reductoisomerase containing the 4_1 knot is found in e.coli and spinach.
- Ubiquitin hydrolase containing the 5_2 knot is found in humans and yeast.

Understanding protein knots is important for medicine

Good knots

Parkinson's disease seems to be caused by the degradation of ubiquitin hydrolase as it passes through a narrow pore of a proteosome and unfolds.

The 5_2 knot, which is normally in ubiquitin hydrolase, might make it difficult for it to thread through the pore which leads to degradation.



Bad knots

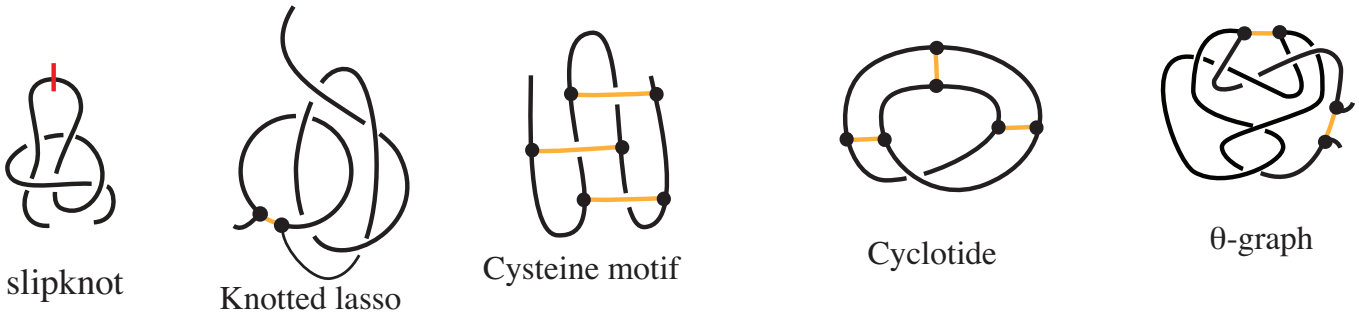
Protein entanglement can cause misfolding diseases including Alzheimer's, oculopharyngeal muscular dystrophy, and certain cancers.

Some members of the SPOUT family of proteins containing a deep trefoil knot are known to cause genetic diseases.

Complex entanglement

Knotting is not the only type of entanglement that can occur in proteins.

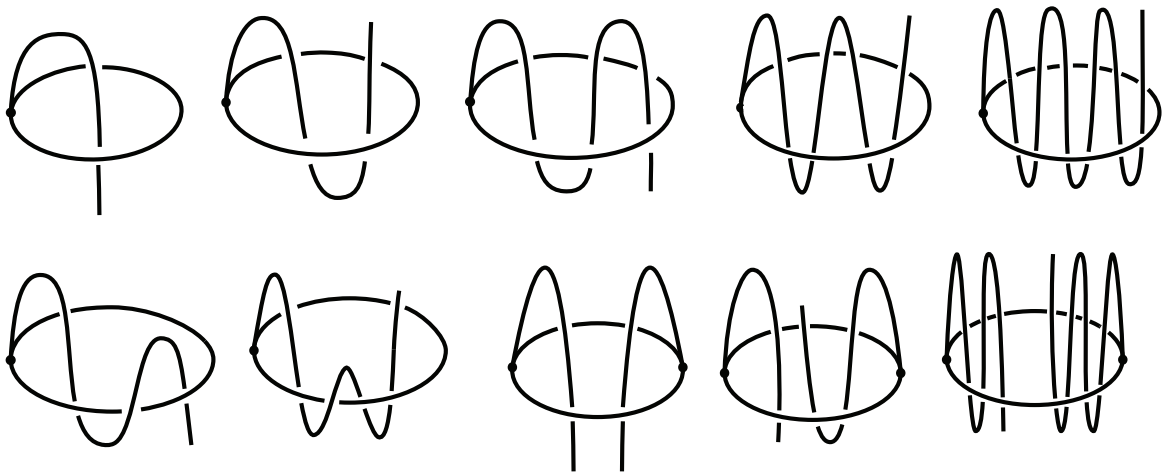
A *slipknot* occurs when a chain is unknotted but it has a knotted subchain.



Lassos have been characterized in terms of how many times they pierce a minimal surface bounded by the loop.

But this doesn't detect knots and other complex entanglements of the tail(s) or loop of the lasso.

Lassos



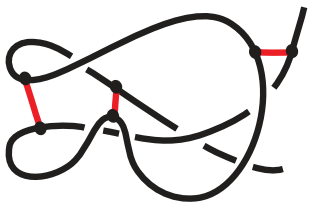
This classification doesn't distinguish more complex lassos.

Also, topologically, these lassos can be undone even while pinning down endpoints, which doesn't occur due to molecular forces.

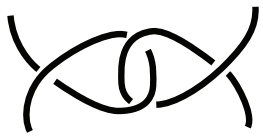
We want a model of entanglement that includes intertwined sites held together by molecular forces and entanglement with intra-chain bonds.

Circuit Topology Model

Alireza Mashaghi introduced *Circuit Topology* in which intra-chain bonds (known as *hard contacts*) and locally entangled units (known as *soft contacts*) are put together with *operations* to create complex entangled protein structures.



hard contacts



held together by
molecular forces

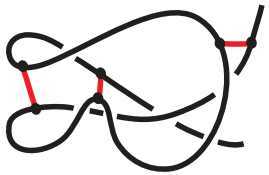
soft contact

Mashaghi found that folding kinetics of polymer chains were correlated with the number and position of hard contacts.

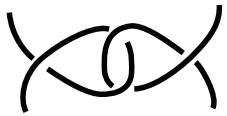
Soft contacts and their operations were not rigorously described.

This is what we are interested in here.

Outline of talk



hard contacts



soft contact

held together by molecular forces

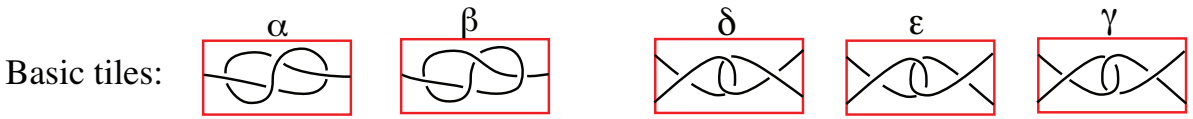
- We model soft contacts and their operations as specific 1-string and 2-string tangles called *tiles*, based on entangled pieces observed by Mashaghi.
- We treat our tiles as rigid because the local entanglement is held together by molecular forces.
- We join tiles together with operations based on the operations for hard contacts developed by Mashaghi.
- We characterize all knots that can be obtained using our tiles and operations.

Basic tiles

A tile is a 1-string or 2-string tangle projected onto a rectangle with compass points NW, SW, SE, and NE indicated or implied.

We start with two 1-string and three 2-string tiles based on entanglements of biopolymers, with compass points implied.

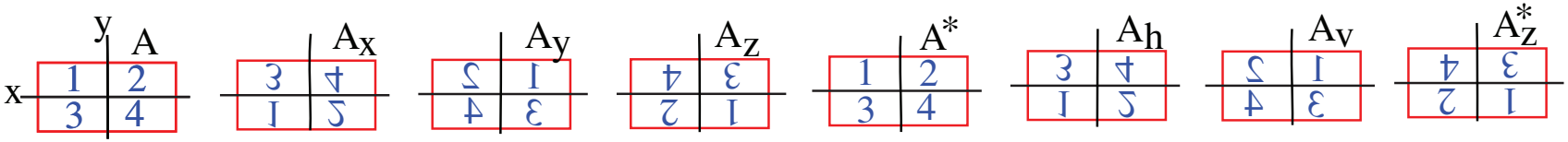
α contains a trefoil knot 3_1, β contains a figure eight knot 4_1, and the other three basic tiles δ, ε, and γ contain *hooks*.



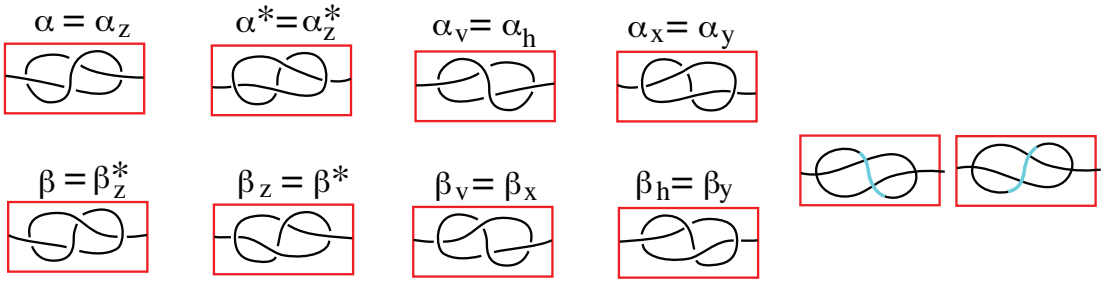
We obtain more tiles by rotating these five basic tiles around long (horizontal) axis x, short (vertical) axis y, and central axis z; and by reflecting tiles across xy-plane, xz-plane, and yz-plane.

Eight 1-string tiles

- A_x is the result of rotating by 180° around (horizontal) x-axis.
- A_y is the result of rotating by 180° around (vertical) y-axis.
- A_z is the result of rotating by 180° around (central) z-axis.
- A^* is the result of a reflection across xy-plane.
- A_h is the result of a horizontal reflection across xz-plane.
- A_v is the result of a vertical reflection across yz-plane.
- A_z^* is the inversion obtained by reflecting A_z across xy-plane.



We obtain the following eight 1-string tiles by rotating and reflecting α and β.

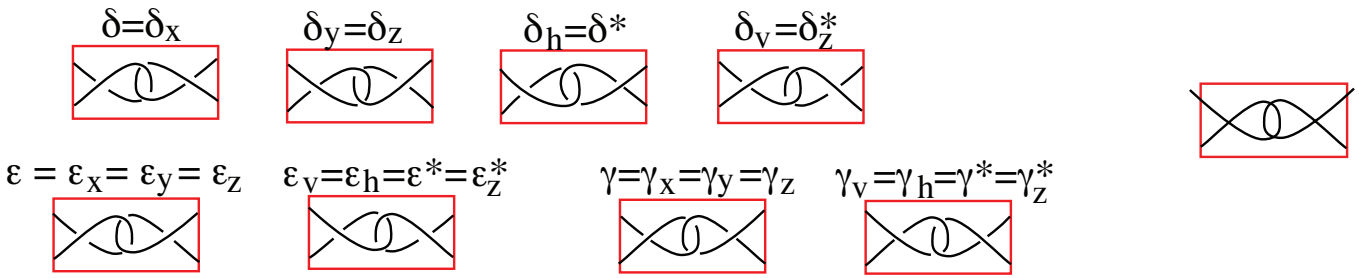


This gives us all non-trivial tiles which have shadows on the right.

Eight 2-string tiles

- A_x is the result of rotating by 180° around (horizontal) x-axis.
- A_y is the result of rotating by 180° around (vertical) y-axis.
- A_z is the result of rotating by 180° around (central) z-axis.
- A^* is the result of a reflection across xy-plane.
- A_h is the result of a horizontal reflection across xz-plane.
- A_v is the result of a vertical reflection across yz-plane.
- A_z^* is the inversion obtained by reflecting A_z across xy-plane.

We obtain the following eight 2-string tiles by rotating and reflecting δ, ε, and γ.

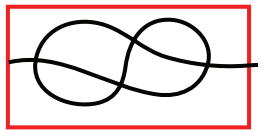


This gives us all non-trivial tiles with shadow on the right.

Three basic tile complexes

We introduce operations to combine any of these 16 tiles to get an entangled arc called a tile complex with all crossings inside the tiles and ends of the complex in the same region of plane.

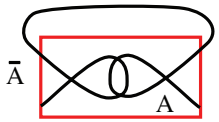
A 1-string tile is already an arc, and hence is already a tile complex.



We don't specify over and under crossings so the drawing can represent any 1-string tile.

The closure operation $\overline{A}$ joins the NE and NW endpoints of a 2-string tile A to make it into a tile complex.

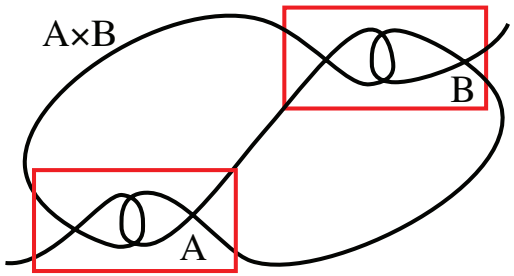
The closure of a 2-string tile



If we join the NE and SE (or NW and SW) endpoints we don't get a single arc.

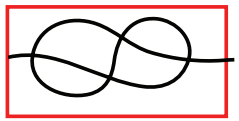
Three basic tile complexes

The cross operation joins 2-string tiles A and B with three arcs to make them into a tile complex $A \times B$.

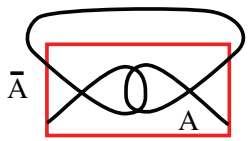


Now in addition to the 16 basic tiles we have three basic tile complexes.

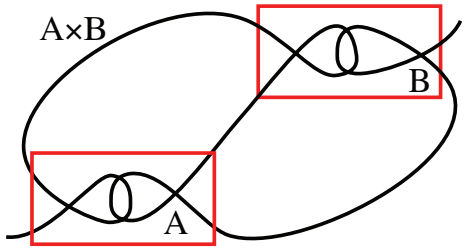
Note, there are other ways we could define this but they give equivalent tile complexes.



1-string tile



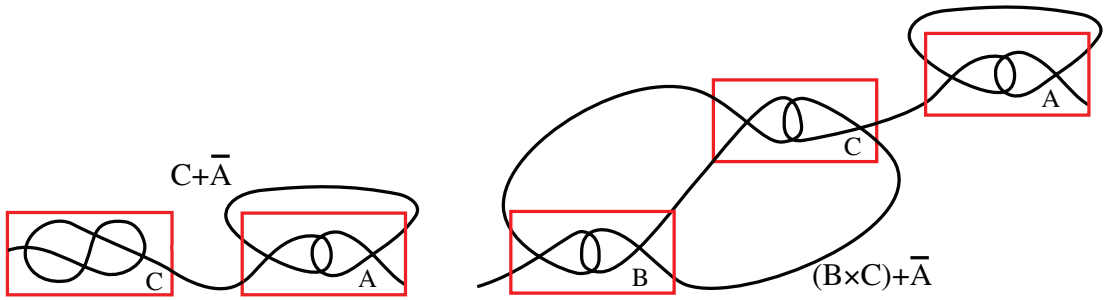
The closure of a 2-string tile



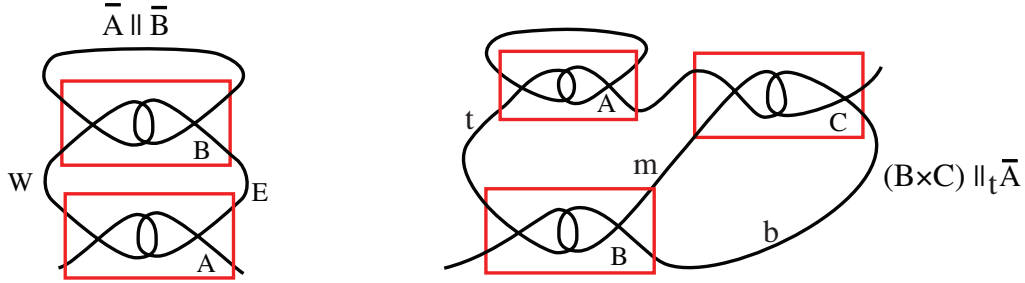
cross

Next we combine tile complexes with two operations

The series operation denoted $S + T$ strings together tile complexes S and T. Note that $S + T$ is not the same as $T + S$.

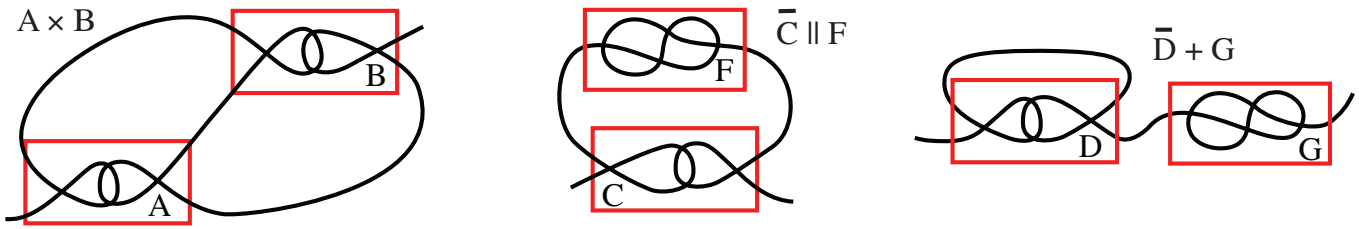


The parallel operation denoted $S \parallel T$ inserts a tile complex T into an arc of a tile complex S which is outside of a tile but not part of a terminal arc. Inserting a tile complex into a terminal arc is series.

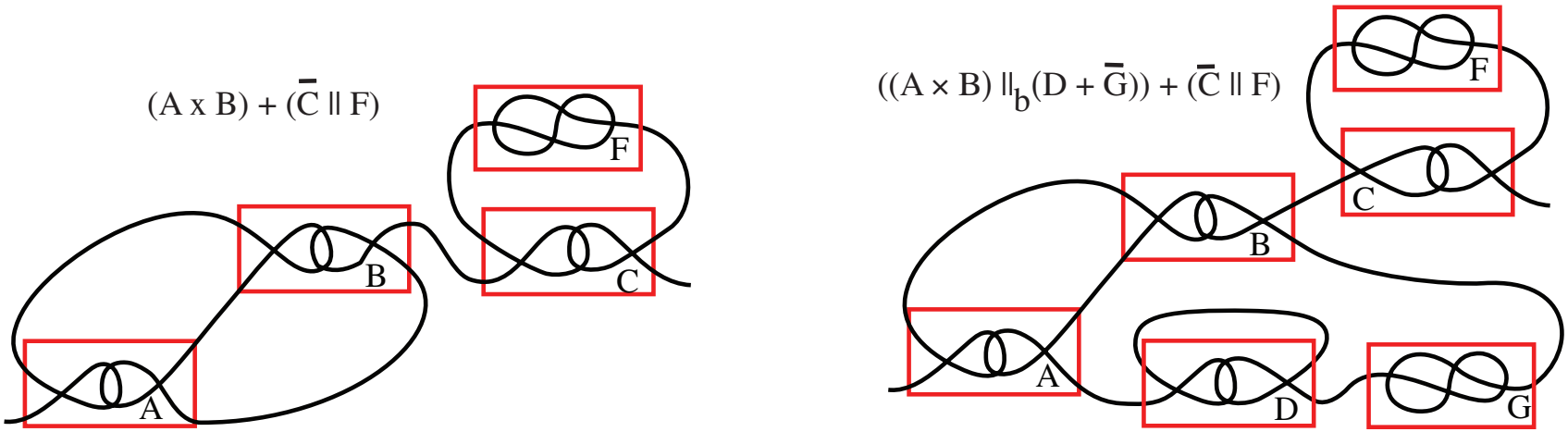


We use subscripts for top (t), middle (m), or bottom (b) when inserting into $\times$, and West (W) or East (E) when inserting into $\parallel$.

A complicated example



Starting with $A \times B$, we add $\overline{C} \parallel F$ on the right end, and insert $\overline{D} + G$ into the bottom string to get the following.



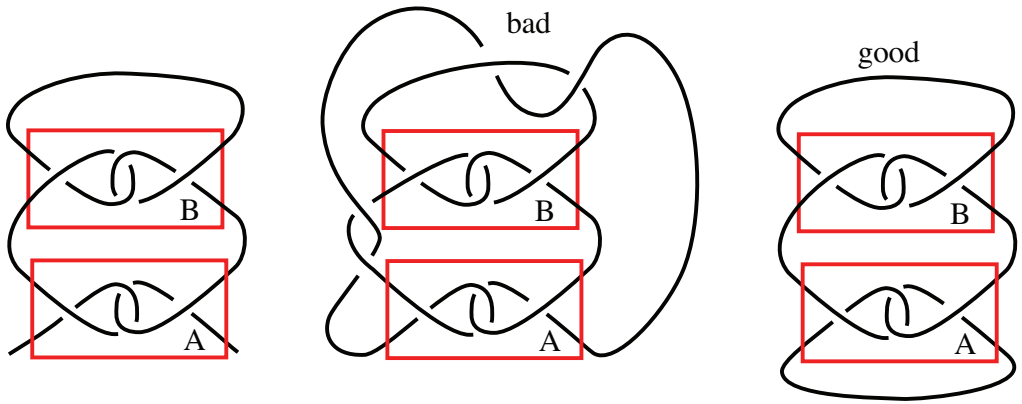
All tile complexes are obtained by combining the three basic tile complexes using *series* (which strings them together) and *parallel* (which inserts one into an interior arc of another).

Making tile complexes into knots

Since all crossings are within tiles and we are treating tiles as rigid, the tangling of a tile complex is trapped.

We join the endpoints of a tile complex the plane to determine the knots we get up to isotopy.

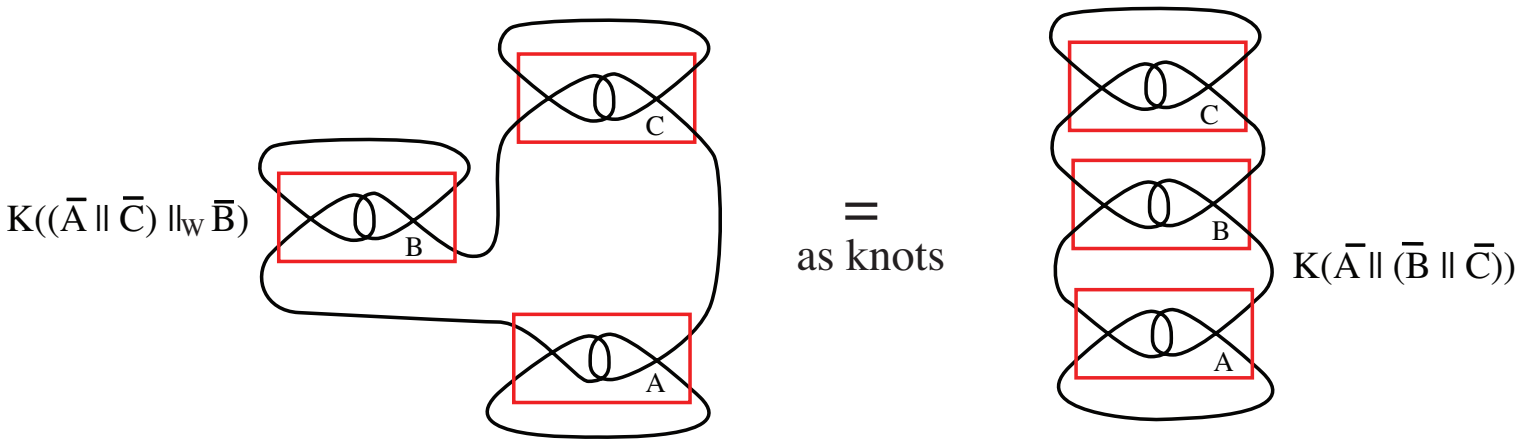
If we join endpoints in space we could introduce tangling outside of the tiles. So we don't do this.



The sealing $K(T)$ of a tile complex T is the knot obtained by joining the endpoints in the plane without adding new crossings.

Our operations leave the endpoints of a tile complex in the same region of the plane, so there is a unique way to join them.

Sealing tile complexes



As knots, these are equal because in $K((\overline{A} \parallel \overline{C}) \parallel_W \overline{B})$ we can slide $\overline{C}$ along an arc of $\overline{B}$ to the top of $\overline{B}$.

But, as tile complexes $(\overline{A} \parallel \overline{C}) \parallel_W \overline{B}$ and $\overline{A} \parallel (\overline{B} \parallel \overline{C})$ are not equivalent, since no isotopy of the plane treating each tile as rigid, takes one to the other.

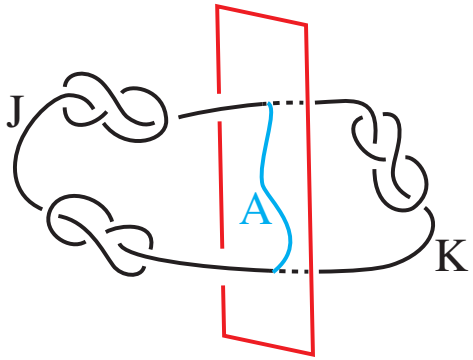
We consider sealings of tile complexes up to isotopy of space to determine all knot types we can get from our model.

Note this does not preserve the tile operations as we see above.

Connected sums

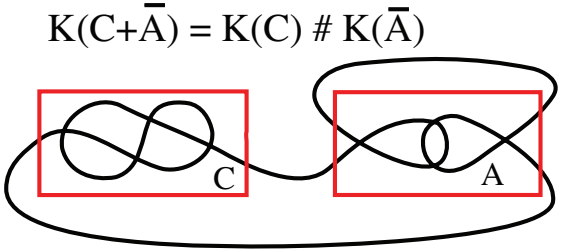
A knot is a *connected sum* $J\#K$ if a plane can split it into non-trivial knots J and K.

J # K

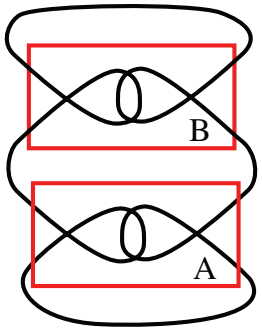


This knot can be split into three non-trivial knots.

Examples of sealings of tile complexes which are connected sums:



Sealing of series (i.e., stringing two tile complexes together)



K(Ā ‖ B̄) =K(Ā) # K(B̄)

Sealing of parallel (i.e., inserting one tile complex into another)

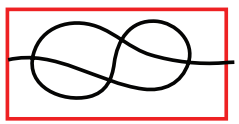
In fact, sealing of series (i.e., stringing two tile complexes together) or parallel (i..e., inserting one tile complex into another) always produces a connected sum.

Lemma

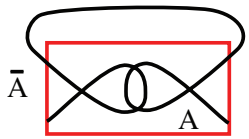
The sealing of a tile complex with at least two tiles is a connected sum where each summand is the sealing of one of the three basic tile complexes.

To determine the knots we can obtain with tiles we only need to know what knots the sealings of the three basic tile complexes are.

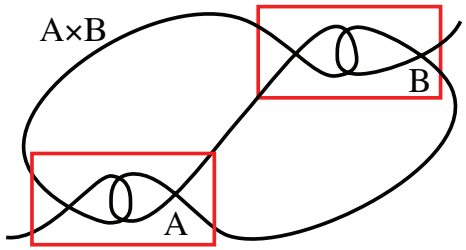
Basic tile complexes:



1-string tile



The closure of a 2-string tile



cross

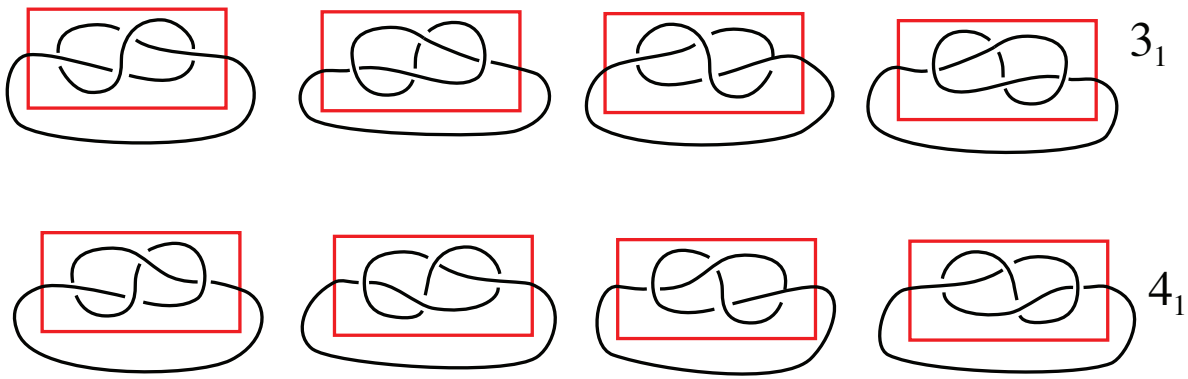
Sealings of 1-string tiles

Recall basic tile complexes are: Eight 1-string tiles, closure of eight 2-string tiles, and the cross of two of the eight 2-string tiles.

Below we determine all sealings of basic tile complexes.

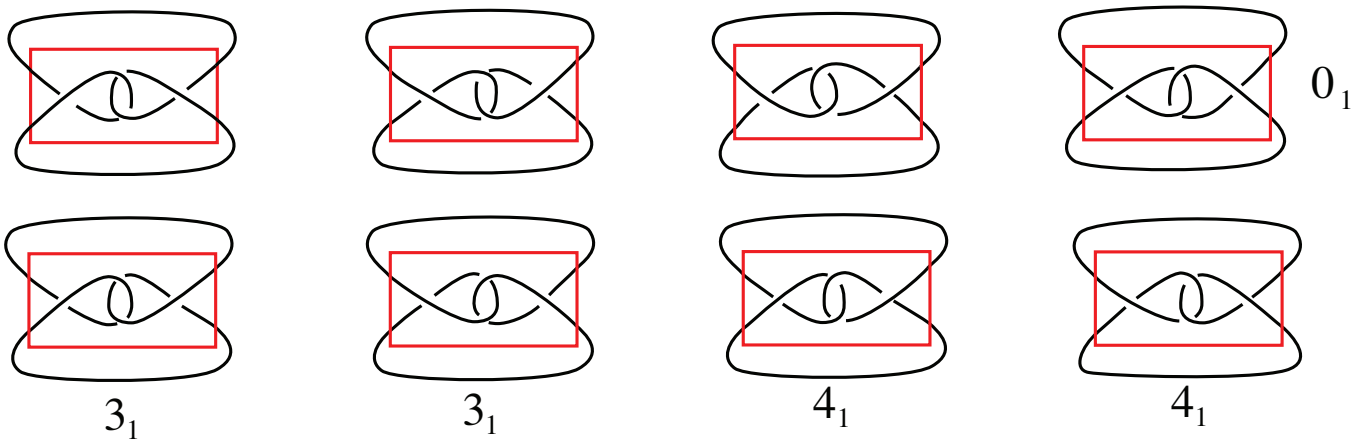
1-string Lemma

The sealing of a 1-string tile is one of the knots $\pm 3_1$ or 4_1.



2-string Lemma

If B is a 2-string tile, then $K(\overline{B})$ is the trivial knot 0_1 or one of the knots $\pm 3_1$ or 4_1.



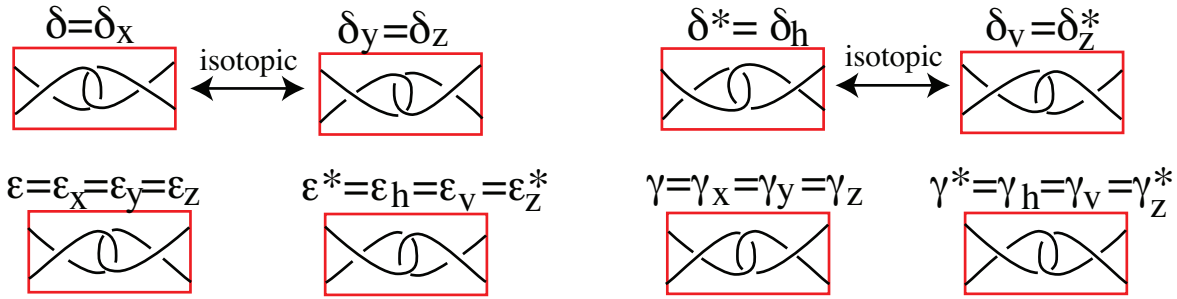
It's harder to determine all sealings of the cross of two 2-string tiles because there are 64 combinations of the eight 2-string tiles, so first we prove a technical lemma.

Technical Lemma

Let A and B be 2-string tiles.

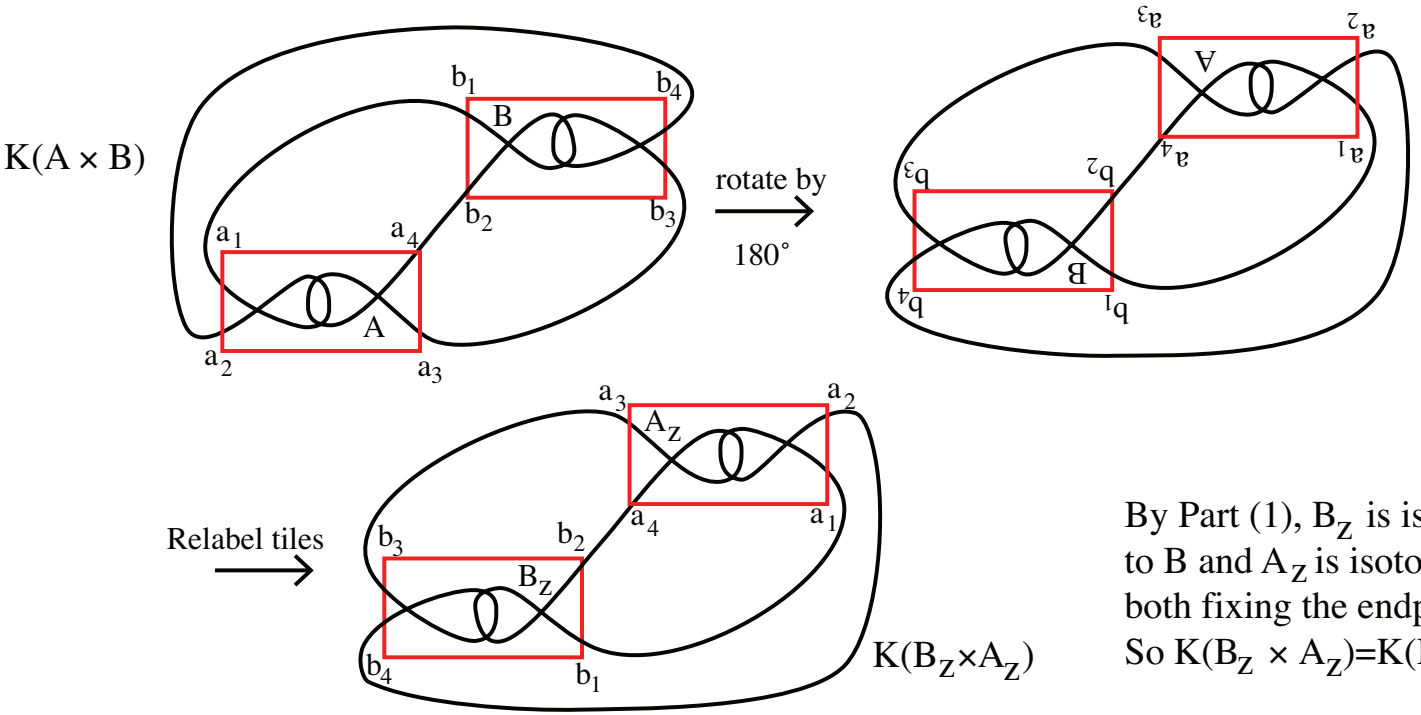
1. Either $A = A_z$ or there is an isotopy of space taking A to A_z fixing the endpoints of the arcs.
2. The only 2-string tiles, up to an isotopy in space fixing the endpoints of the arcs, are: δ, δ^*, ε, ε^*, γ, γ^*
3. $K(A \times B) = K(B \times A)$.
4. $(A \times B)^* = A^* \times B^*$.

Proof of 1 and 2) Up to isotopy of space fixing the endpoints, there are six 2-string tile projections.



Knot sealings of the cross of 2-string tiles

Proof of 3) $K(A \times B) = K(B \times A)$.



By Part (1), B_Z is isotopic or equal to B and A_Z is isotopic or equal to A both fixing the endpoints of the arcs. So $K(B_Z \times A_Z)=K(B \times A)$

Proof of 4) $(A \times B)^* = A^* \times B^*$

Switching all of the crossings of A and B is the same as switching all of the crossings of $A \times B$. $\square$

Technical Lemma

Let A and B be 2-string tiles.

1. Either $A = A_z$ or there is an isotopy of space taking A to A_z fixing the endpoints of the arcs
2. The only 2-string tiles, up to an isotopy in space fixing the endpoints of the arcs, are: δ, δ^*, ε, ε^*, γ, γ^*.
3. $K(A \times B) = K(B \times A)$.
4. $(A \times B)^* = A^* \times B^*$.

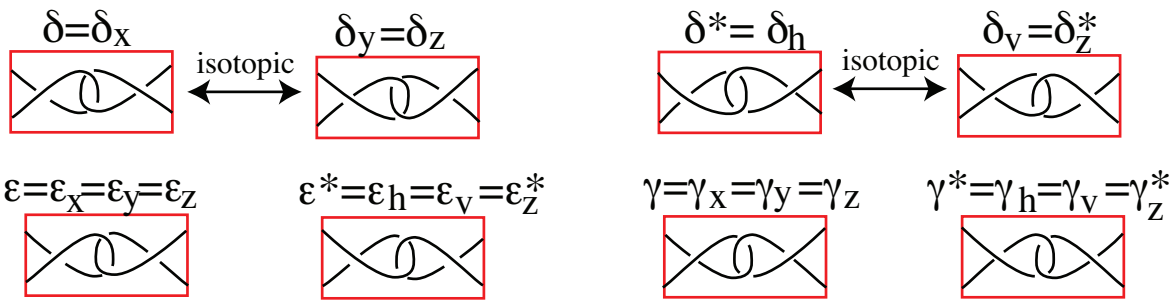
We don't compute mirror image knots. So by (2), (3), and (4), we only compute 12 knot types of $K(A \times B)$ where A is δ, ε, or γ.

$K(\delta \times \delta)$, $K(\delta \times \delta^*)$, $K(\delta \times \varepsilon)$, $K(\delta \times \varepsilon^*)$,$K(\delta \times \gamma)$, $K(\delta \times \gamma^*)$

$K(\varepsilon \times \varepsilon)$, $K(\varepsilon \times \varepsilon^*)$, $K(\varepsilon \times \gamma)$, $K(\varepsilon \times \gamma^*)$

$K(\gamma \times \gamma)$, $K(\gamma \times \gamma^*)$

Knot types of $K(A \times B)$



The table below lists the 12 knots obtained as $K(A \times B)$, which we determined by hand.

cross of 2-string tiles	knot type	cross of 2-string tiles	knot type
$K(\delta \times \delta)$	$+3_1$	$K(\delta \times \delta^*)$	4_1
$K(\delta \times \varepsilon)$	$+5_2$	$K(\delta \times \varepsilon^*)$	0_1
$K(\delta \times \gamma)$	6_1	$K(\delta \times \gamma^*)$	0_1
$K(\varepsilon \times \varepsilon)$	5_1	$K(\varepsilon \times \varepsilon^*)$	6_3
$K(\varepsilon \times \gamma)$	6_2	$K(\varepsilon \times \gamma^*)$	6_1
$K(\gamma \times \gamma)$	7_7	$K(\gamma \times \gamma^*)$	8_{12}

Summary

Lemma

If A is a 1-string tile, then $K(A)$ is 3_1 or 4_1 or their mirror images.

Lemma

If B is a 2-string tile, then $K(\overline{B})$ is 0_1, 3_1, or 4_1.

Lemma

If A and B are 2-string tiles, then $K(A \times B)$ is 0_1, 3_1, 4_1, 5_1, 5_2, 6_1, 6_3, 7_7, 8_{12}.

Theorem

The sealing of any tile complex is a connected sum of 0_1, 3_1, 4_1, 5_1, 5_2, 6_1, 6_3, 7_7, 8_{12}.

Until recent predictions by machine learning program AlphaFold, the above list included all known protein knots.

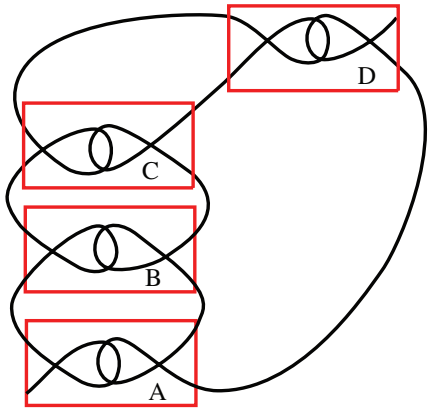
Sequence Notation

Above we wrote symbolic expressions to describe how knotted chain can be constructed using tiles and operations.

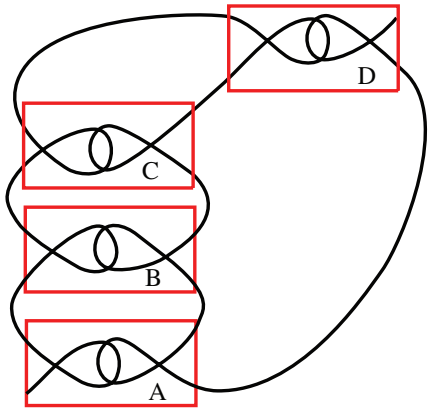
Example: $(\overline{A} \parallel \overline{C}) \parallel_W \overline{B}$, describes how to construct a knotted chain starting with particular 2-string tiles A, B, and C.

However, we would also like to start with a projection of a knotted chain and write it as a tile complex to see how to construct it with tiles and operations.

Example: Can the following be constructed as a tile complex?



Sequence Notation



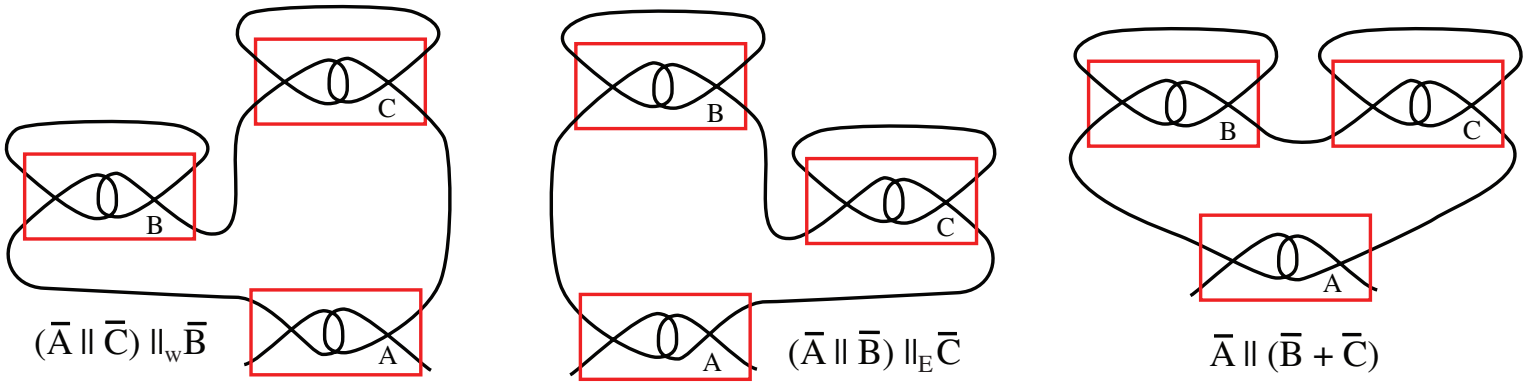
As a first step, we introduce *Sequence notation* (based on Dowker–Thistlethwaite notation for knots) which is read off from a tangled arc projection going from the W endpoint to the E endpoint.

The above projection has sequence notation *ABCDCBAD*.

Note that just because it can be written as a sequence of tiles does not mean it can be constructed using our operations.

Sequence Notation

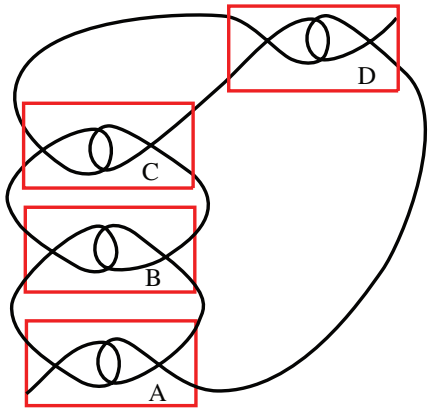
These are equivalent tiles complexes (because their projections are the same), and have sequence notation *ABBCCA*, but their operation notations and their constructions are distinct.



Rules for a sequence to represent a tile complex

1. Each letter either appears once and represents a 1-string tile or appears twice and represents a 2-string tile.
2. At most one letter can alternate with a given letter.

Rule 2) At most one letter can alternate with a given letter.



ABCDCBAD violates Rule 2 since *A*, *B*, and *C* all alternate with *D*. This sequence does not represent a tile complex.

Theorem

Any sequence which satisfies Rules 1 and 2 represents a tile complex.

We prove this theorem below by describing an algorithm going from a sequence satisfying Rules 1 and 2 to a tile complex.

Definition

A letter appearing only once in a sequence is called a singleton, two consecutive instances of the same letter such as AA are twins, and adjacent alternating letters such as $ABAB$ is an interweaving.

Example: $ABADDBCC$ has two pairs of twins, but no singleton. It has no interweaving, though A and B alternate.

Lemma

Any sequence with no singletons which satisfies Rules 1 and 2 must either have a pair of twins or an interweaving.

We don't prove this here, but use it to define an algorithm to go from a sequence satisfying Rules 1 and 2 to a picture of a tile complex together with its operation notation.

Algorithm together with example *ABADDEBCFC*

Step 0. Let S_0 denote a sequence satisfying Rules 1 and 2.
Example: $S_0 = ABADDEBCFC$. We check it satisfies the Rules.

1. Each letter either appears once and represents a 1-string tile or appears twice and represents a 2-string tile. E and F are the only letters that appear just once.

2. At most one letter alternates with a given letter.

Step 1. Let S_1 denote the sequence obtained by deleting all of the singletons from S_0. If S_0 has no singletons, then let $S_1 = S_0$.
We delete the singletons E and F from S_0 to obtain $S_1 = ABADDBCC$.

Step 2. If $S_1 = \emptyset$ (i.e., all letters of S_0 were singletons), then go to Step 9. Otherwise, $S_1 \neq \emptyset$ and has no singletons. So by Lemma, either S_1 has a pair of twins or an interweaving.
S_1 contains twins DD and CC, but no interweavings.

Algorithm together with example *ABADDEBCFC*

Step 3. If $S_2 = \emptyset$, then go to Step 5. Otherwise, by Lemma, S_1 either has a pair of twins or an interweaving. S_2 is obtained by deleting the twins and interweavings from S_1.
We delete twins DD and CC from $S_1 = ABADDBCC$ to get $S_2 = ABAB$.

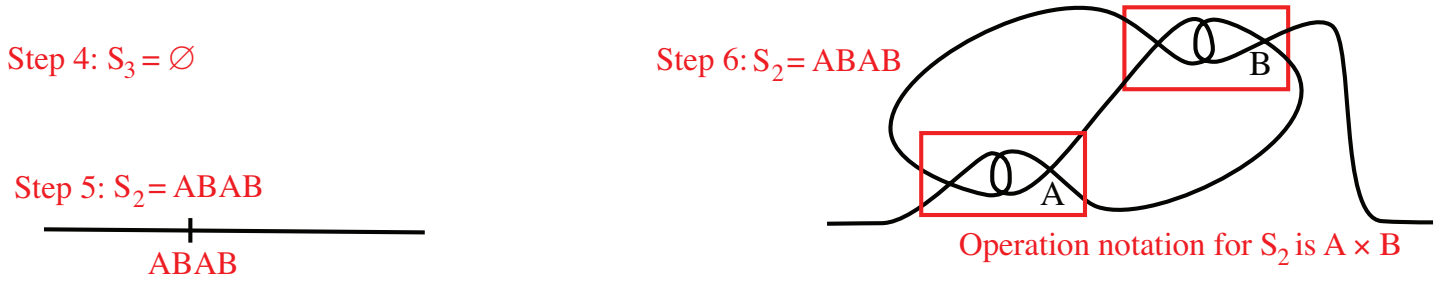
Step 4. Repeat Step 3 until for some n, we have $S_n = \emptyset$.
We delete $ABAB$ from S_2 to get $S_3 = \emptyset$.

Step 5. If $S_1 = \emptyset$, we skipped to Step 9. Thus $S_{n-1} \neq S_0$ and hence S_{n-1} contains no singletons. Represent S_{n-1} as a line segment with markings for twins and interweavings in the order they occur in S_{n-1}.

$S_{n-1} = S_2 = ABAB$

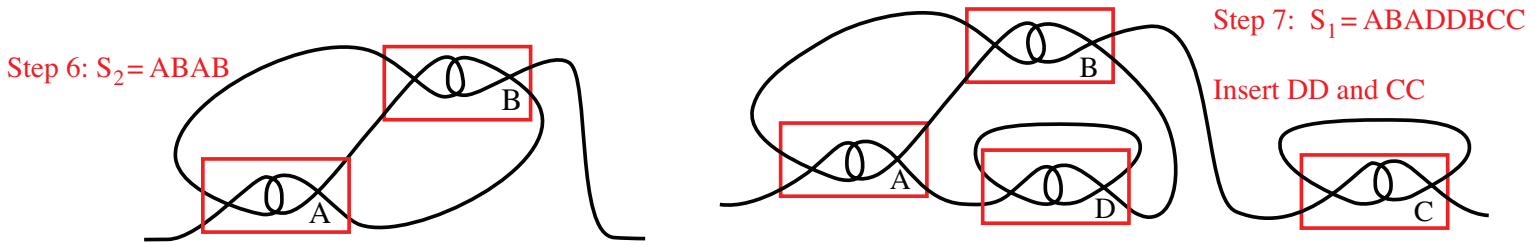
Algorithm together with example *ABADDEBCFC*

Step 6. Replace markings for twins and interweavings by pictures to get a tile complex with sequence S_i, where $i = n - 1$.



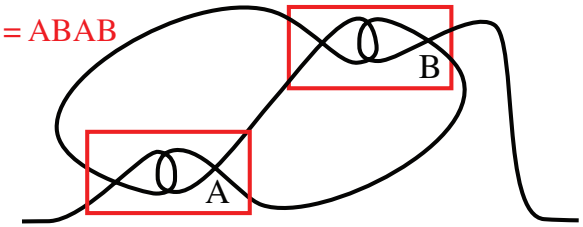
Operation notation for S_i: Because S_i is obtained by inserting twins and interweavings into a line segment, S_i is the series operation with summands of the form $\overline{C}$ or $A \times B$.

Step 7. If $i = 1$, go to Step 8. Otherwise, insert twins and interweavings from the sequence S_{i-1} into the tile complex for S_i to get a tile complex for S_{i-1}.



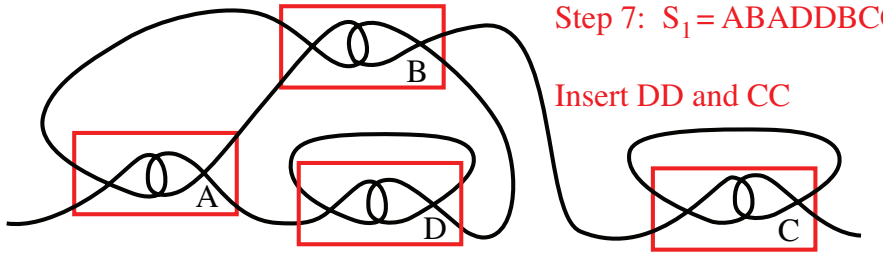
Algorithm together with example *ABADDEBCFC*

Step 6: S_2 = ABAB



Step 7: S_1 = ABADDBCC

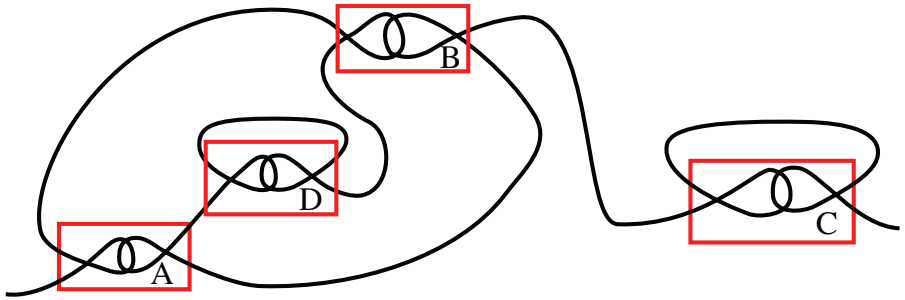
Insert DD and CC



Note that the position of the DD in the sequence tells us where to insert $\overline{D}$.

The following is incorrect:

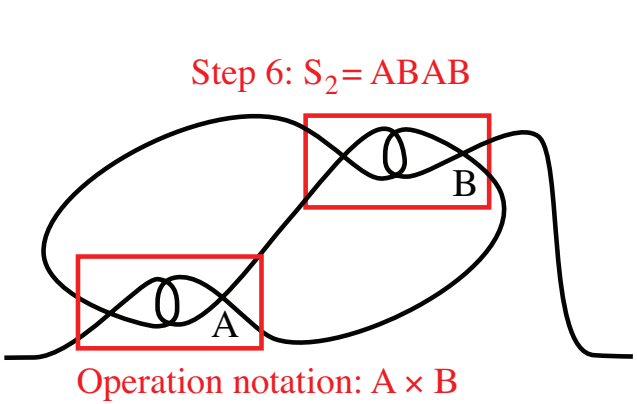
This is ABDDABCC

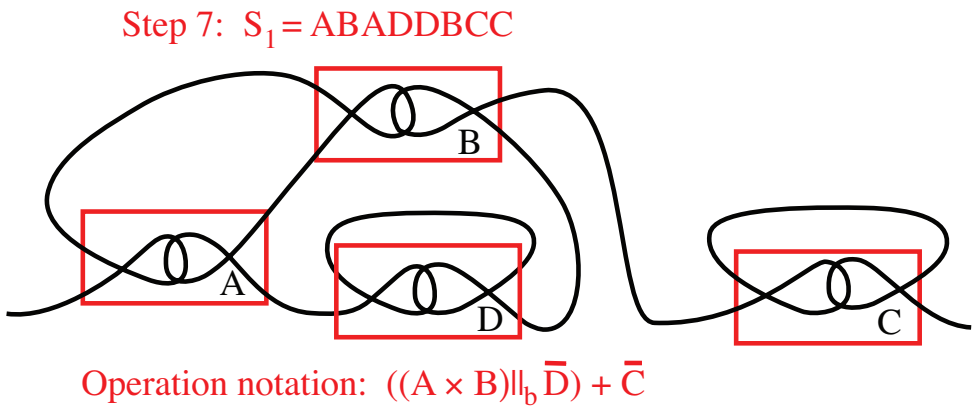


Algorithm together with example *ABADDEBCFC*

Operation notation for S_{i-1}: Starting with the operation notation for S_i, we use series and parallel to insert twins and interweavings into the notation according to the picture.

Operation notation for $S_2 = ABAB$ is $A \times B$. So from the picture we see that operation notation for $S_1 = ABADDBCC$ is $(A \times B) \parallel_b \overline{D}) + \overline{C}$.



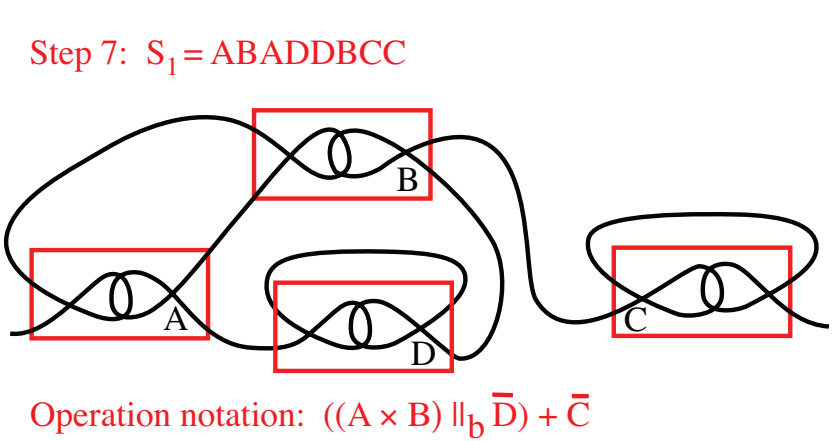


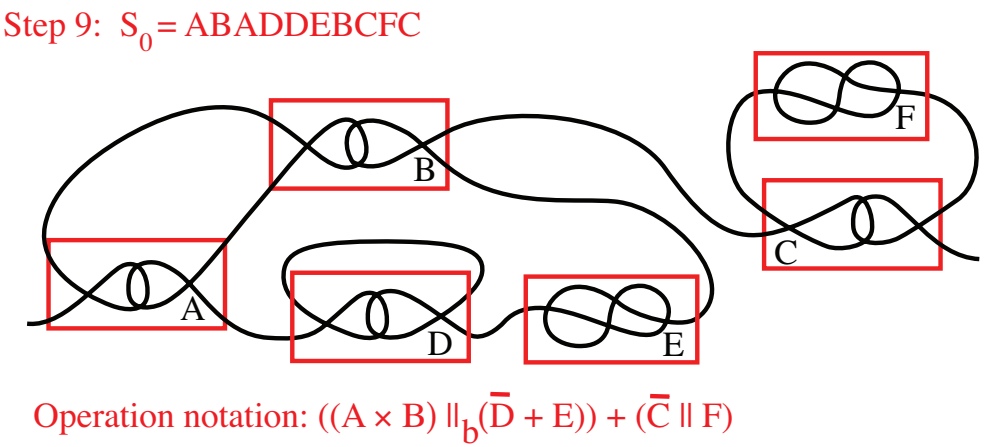
Algorithm together with example *ABADDEBCFC*

Step 8. Repeat Step 7, for each $i < n - 1$ until we get a tile complex for S_1.

In our example, we have a tile complex for S_1 after Step 7, so we skip Step 8.

Step 9. Insert any 1-string tiles that were singletons in S_0.



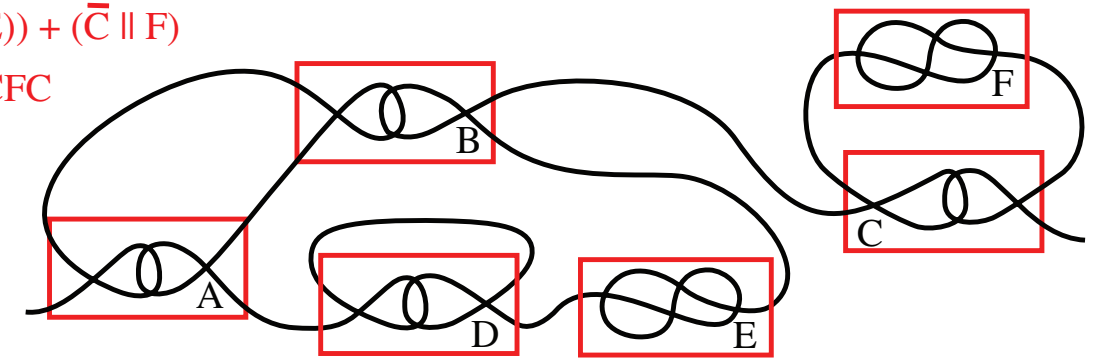


Operation notation for S_0: Use series and parallel to add 1-string tiles to the operation notation for S_1 to get operation notation for S_0.

Comparing notations

Operation notation: $((A \times B) \|_b (\bar{D} + E)) + (\bar{C} \| F)$

Sequence notation: $S_0 = ABADDEBCFC$



We use operation notation to describe how a tile complex is constructed.

We use sequence notation to describe a projection made up of tiles.

Given only a sequence which satisfies Rules 1 and 2, we use our algorithm to obtain a projection of a tile complex and its operation notation.

Intra-chain bonds in our model

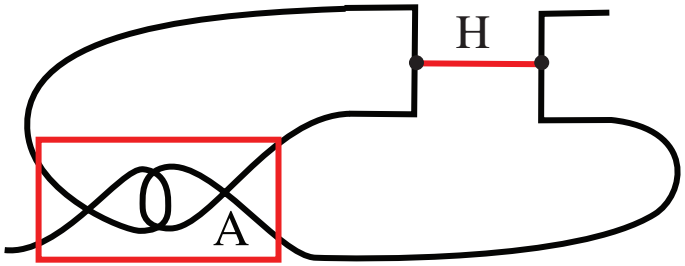
In order to include intra-chain bonds in our tile model we introduce *H-joints*.

Since a protein chain can have some flexibility around intra-chain bonds, we allow the black edges to rotate around the red arc.

Thus an H-joint is not a rigid tile.

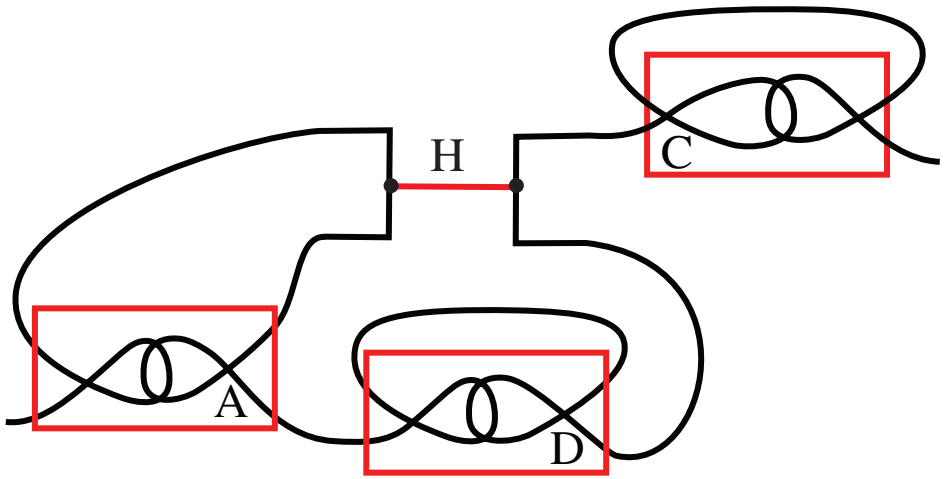
We can now build entangled graphs by using our operations to combine tiles and H-joints, and use our notations to represent them.

Example



Operation notation: $A \times H$
Sequence notation: AHAH

Operation notation: $((A \times H) \|_b \bar{D}) + \bar{C}$
Sequence notation: AHADDHCC



T H A N K
Y O U