Hi David,

Here’s my talk on knots! (12:35 am, goal = write 3 pages of text today)

So I want to convince you that (12:50 am)

TODO = section; ### = gap; @@@ = badly phrased; +++ = might need to be moved, \*\*\* = diagram

[TODO thesis]

[TODO phone]

[TODO intro to knot theory, applications]

Shoelace -> Tie ends together [3] -> Trefoil (isotopy)

[TODO Dowker notation]

augment with information

notation for a knot:

knot for a notation:

dealing with links, duplicates

[TODO enumerate with Dowker? should this go here?]

[TODO enumeration/tabulation + history of + equivalence]

prime alternating knots of small crossing numbers, ignoring mirror images

There are two main schools of knot notation, plus some other interesting notations I won’t mention. One school wants to take an existing knot and break it down. From least recent to most recent, this school includes Gauss, Dowker, and Hoste. Another school wants to define “irreducible” bits of a knot and use them to build up a knot. This school includes Kirkman, Conway, and Caudron.

We talked about Dowker notation and how it breaks down a knot by tracing around it. To draw a knot from Dowker notation, we need to build up the knot, and it might not be unique. @@@ Now let’s talk about how we can define primitives with which to build up a knot.

So, let’s talk about Conway notation!

notation for a knot:

knot for a notation:

The other way: to write a Conway notation from a knot, we need to break down the knot into the primitives, and it may not be unique; but on the other hand, drawing the knot is unique and easy!

As a hook, take this tantalizing quote from Conway’s seminal paper: "Little tells us that the enumeration of the 54 knots of [6] took him 6 years -- from 1893 to 1899 -- the notation we shall soon describe made this just one afternoon's work!"

And this quote: “The enumeration process is eminently suitable for machine computation ... recent attempts at computer enumeration have proved unsatisfactory mainly because of the lack of a suitable notation.”

How did Conway accomplish this? Namely, how did he invent a **notation** or **representation** of knots that enabled him to enumerate them so much more easily and efficiently? Let’s find out! First let’s talk about the primitives that he defined, which are twists; what operations we can do on these primitives to make them become tangles, and what we can do to tangles to make them become knots.

A tangle

[cite Piponi, tangle dance]

A tangle is built by starting with two ropes with ends in the four corners of a circle: \*\*\*

They’re either the 0 or infinity tangle.

Then, we twist the tangle either once upward or once downward.

Tangle as datatype:

0 | inf | twist up | twist down | add tangle tangle | mult tangle tangle | (reflect?)

(3 is shorthand for adding 1 1 1)

We add two tangles by simply smashing them together. \*\*\*

We multiply two tangles by reflecting the first one over the northwest-southeast diagonal, then adding it to the second one. The reflection can also be done by reversing all the crossings (undercrossing to overcrossing and vice versa) and rotating the tangle by 90 degrees clockwise. \*\*\*

We can twist a tangle multiple times by adding multiple 1 tangles. So, we’ll use, say, “3” as a synonym for “1 + 1 + 1”.

### 1 1 1 ... 1 is not in general the same as n, right

Footnote on DNA (cite Chicago paper)

Now we have this twisty thing – how do we make it into a knot? Well, as we said earlier, we have these (### irreducible) components. We pick a structure called a polyhedron and insert these tangles into the structure. (TODO won’t elaborate more on this)

Notice that the resulting knots are always alternating (### and reduced). The tangle is (reduced?).

If we wanted to enumerate all knots up to 5 crossings, how would we do that? Well, notice that each twist introduces exactly one crossing, and every operation combining tangles adds the number of crossings in the tangle. Therefore, find all ways to divide the desired number of crossings into separate numbers – all partitions of n. Each partition corresponds to a rational ### knot with that number of crossings! It’s very straightforward!

pictures: 3 1 1 has 5 crossings...

(equation here)

Reidenmeister moves

flype

Tait’s theorem (and prime alternating knots)

concretely enumerating with code

knots to <= 7 crossings

Two tangles are equivalent if one can be transformed to the other by performing a series of Reidemeister moves on them to transform them into the same tangle, while keeping the ends fixed. One consequence is that, since the ends are fixed, if two knots contain tangles that are all equivalent, the knots are equivalent!

[cite the guy’s paper on converting knots to tangles]

So, this goes a long way toward helping us check knot equivalence, which is hard and unsolved. @@@ Conway proved a powerful theorem about tangle equivalence: two rational tangles are equivalent if and only if they are represented by the same rational number! This rational number is calculated by taking the tangle’s notation and converting it backward into a continued fraction.

checking equivalence via flypes

flypes preserve crossing number

### what’s going on with the flypes? (3, 2, 1, 1) = 7? why write it out? do things under flyping have the same rational fraction? why are we using comma notation? what’s standard form?

is conway’s claim true??

dealing with higher-crossing knots, duplicates, links, polyhedra, other operations

(TODO refer to knot book, linknot, etc.)

[TODO computers vs. humans. knots of large crossing number vs. small crossing number]

quote Hoste on Conway (large number of primitives) vs what they used, Dowker (both him and Thistlewaite, but needed ad-hoc checks)

[TODO comparison]

Breaking down objects vs. building them up

Notation to object correspondence: one-to-many, many-to-one

Human readability/writability

Machine readability/writability

Simplicity (number of symbols and functions)

Ease of performing operations on the represented object

Ease of reading properties off notation

Insights it encodes

New manipulations and ways of thinking it encourages

[TODO lessons]

Blob of thought in head <-> knot

Programming language <-> notation or representation

Program <-> representation for that knot

Reduce the problem to cases + special cases, often relying on powerful theorems

Encode insights in your notation, often relying on powerful theorems

A notation has trade-offs along several axes -- sacrifices in one area can lead to big gains in another

Best,

Katherine