Random Knot Diagrams through 10 crossings

Jason Cantarella, Harrison Chapman* and Matt Mastin†

Keywords:

1. INTRODUCTION

There is currently a great deal of interest in the topic of random geometry and topology. The study of random knots goes back to the 1960's, when polymer physicists realized that the knot type of a closed (or ring) polymer would play an important role in the statistical mechanics of the polymer [**?**]. Orlandini and Whittington give a comprehensive survey of the development of the field since [14]. In this area, a random knot is usually modeled by a closed random walk, often on the integer lattice. For almost all of these models, certain phenomena have been rigorously established– for instance, the probability of knotting goes to 1 exponentially fast as the size of the walk increases. However, it has been difficult to prove more informative theorems.

One way to look at the problem is that it is quite hard to relate the three-dimensional shape of a space curve to its knot type; one can in principle express the finite-type invariants as (complicated) integrals in the spirit of [11], but so far computing the expected value of these integrals has been too difficult.

The point of this paper is to look at the set of knot diagrams not as convenient combinatorial representations of the shape of space curves, but as a probability space in their own right. This is basically the same model of random knotting as [6] or [5]: the objects are equivalence classes of immersions of S^1 into S^2 as in Arnol'd [1] paired with points on the hypercube $\{\pm 1\}^{n+1}$ representing assignment of global orientation and crossing signs. There are other quite different combinatorial models of random knotting in the literature– see [4], [7], or [13].

Previous authors have sampled a slightly different version of this space using an algorithm of Schaeffer and Zinn-Justin [15] to uniformly sample a space of marked link diagrams and keeping only the knots. By contrast, we carried out an complete enumeration of composite knot diagrams up to 11 crossings using the graph theory software *plantri* [2, 3]. We checked our results by using *plantri* in two different ways to generate our graphs, as described below.

This paper is primarily computational. While we give a proof that our algorithm for enumerating diagrams is correct, our main contribution is the data set of diagrams itself. As we mentioned

*University of Georgia, Mathematics Department, Athens GA
†Wake Forest University, Mathematics Department, Athens GA

above, Arnol'd's seminar studied the immersions of the circle into the plane in the 1990's as part of his general theory of finite type invariants of plane curves. Arnol'd's main paper provides an explicit count of the same immersions of S^1 into S^2 that we enumerate, but only for $n \leq 5$ [1, page 79]. This is sequence A008989 in the Online Encyclopedia of Integer Sequences [**?**], with an extension to $n = 7$ credited to Guy H. Valette. We could not find a published reference for Vallette's extension of the table, but our data for knots below matches A008989 including the extension, and extends the sequence further to $n = 11$.

How many of these diagrams represent nontrivial knots? The fraction of unknots is very high for small crossing numbers and begins to decay as the number of crossings in the diagram increases. For some diagrams (called "tree-like" curves), the knot type does not depend on the assignment of crossing information– the resulting knot is *always* the unknot (see Figure **??**). These diagrams turn out be surprisingly common for low crossing numbers. For example, 42.05% of 8-crossing diagrams are treelike, which explains about half of the unknots among 8-crossing diagrams (84% of 8-crossing diagrams are unknots). The remaining unknots are almost all connect sums of treelike diagrams with a single trefoil or figure-8 diagram. We will show that a simple analysis based on such diagrams gives a lower bound of 77% for the unknot fraction in 8-crossing diagrams, explaining more than 90% of all unknots in this class.

Motivated by this, we then focus on the connect sum structure of random diagrams. This is extremely interesting, as our data shows that the vast majority of diagrams are highly composite. This is partially explained by the existence of pattern theorem for diagrams [**?**], but more work remains to characterize the distribution of various sizes of summands in a random diagram.

Counting knot diagrams is made significantly more complicated by the existence of diagram symmetries; for instance, a correction factor due to symmetries is the main technical difference between this natural model and the model of [15]. Our data shows that this difficulty rapidly disappears as the number of crossings increases; the average size of the automorphism group for a random 10 crossing knot diagram, for instance, is $846929/823832 \simeq 1.028$.

2. DEFINITIONS

We begin with some definitions.

Definition 1. We define a *link shadow* L with n vertices to be an equivalence class of generic smooth immersions of a collection of oriented circles into S^2 with n intersections up to diffeomorphism of the sphere (which may not be orientation-preserving).

Examples of link shadows are shown in Figure 1. In our experiments, each link shadow will be represented (non-uniquely) by a combinatorial object called a PD-code.

Definition 2. A **PD-code** $\mathfrak{L}$ with n-vertices is a list of n cyclically ordered quadruples called vertices which partition the set of signed edges $\pm 1, \ldots, \pm 2n$ so that in each quadruple every pair of

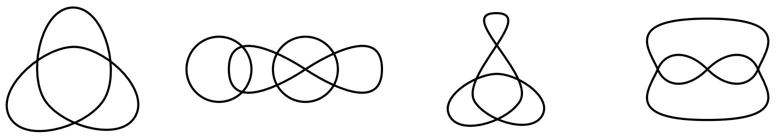


FIG. 1: Examples of link shadows.

non-adjacent edges have opposite signs. Two pd-codes are pd-isomorphic if there exists a bijection of vertices and edges which respects the partition of signed edges into vertices, including cyclic ordering.

For example $\mathfrak{L}_0 = ((+4, -2, -5, +1), (+2, -6, -3, +5), (+6, -4, -1, +3))$ is a valid pd-code of three vertices and six edges, as each of $\pm 1, \ldots, \pm 6$ occurs once, and each pair of non-adjacent labels in a single quadruple (such as $+4, -5$ and $-2, +1$ in the first quadruple) has opposite signs.

Definition 3. Given a pd-code $\mathfrak{L}$ and a signed edge label e, we can define the *successor* of $s(e)$ to be minus the edge immediately preceding e in the cyclic ordering of the quadruple where e occurs. The faces of a pd-code $\mathfrak{L}$ are the orbits of the successor map.

In the pd-code above, for example, -2 occurs in the quadruple $(+4, -2, -5, +1)$, so $s(-2) = -(+4) = -4$. Similarly, $s(s(-2)) = s(-4) = -(+6) = -6$, and $s(s(s(-2)) = s(-6) = -(+2) = -2$. Thus $(-2, -4, -6)$ is a face of $\mathfrak{L}_0$.

Proposition 4 (Mastin)**.** *The vertices, edges, and faces of $\mathfrak{L}$ form a cell-complex structure on a 2-dimensional surface $C(\mathfrak{L})$. Given two pd-codes $\mathfrak{L}$ and $\mathfrak{L}'$, the pd-codes are pd-isomorphic $\iff$ the cell-complexes $C(\mathfrak{L})$ and $C(\mathfrak{L}')$ are isomorphic.*

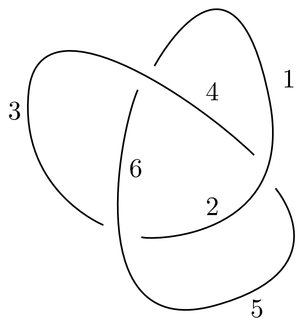
We can then define

Definition 5. The *genus g* of a pd-code $\mathfrak{L}$ is the genus of the associated cell complex $C(\mathfrak{L})$. It is given by $V - E + F = 2 - 2g$, where V, E, and F count the vertices, edges, and faces of $\mathfrak{L}$.

It is another theorem of Mastin that

Proposition 6 (Mastin)**.** *There is a bijection between n-vertex link shadows and n-vertex* genus zero *pd-codes up to pd-isomorphism.*

The bijection itself is easy to construct: the vertices of the pd-code represent intersections of the circles, the edges represent the sections of circles between intersections (signs represent orientation along the circle), and the cyclic ordering of the vertices is the counterclockwise order of circle arcs at each intersection point. An example of a link shadow and the associated PD-code is shown in



$$\{[+4,-2,-5,+1],[+2,-6,-3,+5],\\ [+6,-4,-1,+3]\}$$

FIG. 2: A diagram for 3_1 and its PD-code. The labels are only single integers here as there is only one component. Note that we may omit directional arrows as the orientation can be inferred from the ordering of the edge labels.

Figure 9. It is clear that there are finitely many n-vertex genus zero pd-codes, and so finitely many equivalence classes of pd-codes up to isomorphism. By the proposition, there are therefore finitely many n-vertex link shadows. This will be our initial probability space.

Given a link shadow $\mathfrak{L}$ we can define a link diagram by

Definition 7. A *link diagram* is a link shadow where each intersection is decorated with over-under information for the circles meeting at the intersection. We call these intersections *crossings*. The equivalence relation for diagrams is the *diagram isomorphism*: a diffeomorphism of S^2 to S^2 which respects over-under information.

It is clear that there are at most $2^{\#\text{ crossings}}$ link diagrams associated to a given link shadow, but that this number will be smaller if there are nontrivial shadow automorphisms.

Definition 8. In the *random diagram model*, a random n-crossing knot is selected uniformly from the counting measure on the finite set of one-component n-crossing link diagrams.

This is the smallest probability space including all the knot diagrams that one can define; however, note that we could expand the probability space by doing things like choosing planar, rather than spherical embeddings (hence labeling a face as the exterior) or choosing a basepoint on each component. These amount to making the rules for shadow isomorphism more strict, increasing the number of equivalence classes of diagrams.

We intend to compute knot probabilities directly in this model for (relatively) small n by direct enumeration. Having the entire collection of diagrams will in addition allow us to study transitions between diagrams, but we will discuss this in future work.

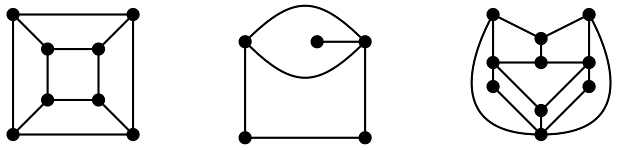


FIG. 3: Examples of quadrangulations.

3. ENUMERATING SHADOWS

Our first goal was to enumerate the link shadows computationally. This section describes our two enumeration algorithms. Each is built on the same computational foundation; a library which allows us to manipulate pd-codes. In this library, we provide

Definition 9. A *pdstor* is a ordered collection of pd-isomorphism types of pd-codes. We define the operation of adding a pd-code to a pdstor as adding the pd-code if it does not belong to an existing pd-isomorphism type (that is, is not pd-isomorphic to a pd-code present in the pdstor) and ignoring the pd-code otherwise.

Our implementation uses a combination of hashing and brute-force comparison to check whether a given pd-code is already pd-isomorphic to something in the database.

Details are provided in Appendix A for the curious reader. We are left with the problem of creating a set of input pd-codes to the pdstor which are guaranteed to cover all n-vertex shadows.

3.1. Dual quadrangulations and connect sums

Brinkmann et al. [2] provide an algorithm to enumerate all of the simple embedded planar quadrangulations of S^2, up to embedded isomorphism. A quadrangulation is a planar graph where each face has four edges, as in Figure 3. The dual graph to a quadrangulation is a connected 4-regular embedded planar multigraph, as shown at left in Figure 4. In other words, the dual graph defines a link shadow. This is almost enough to enumerate shadows. However, not every link shadow is obtained in this way: if the quadrangulation is simple, no pair of faces in the link shadow share more than one edge. As we can see in Figure 4, this property is not true for every link shadow. We need a familiar idea in a slightly new guise: prime and composite shadows.

Definition 10. Every pair of faces in a *prime shadow* shares at most one edge; all other link shadows are *composite shadows*.

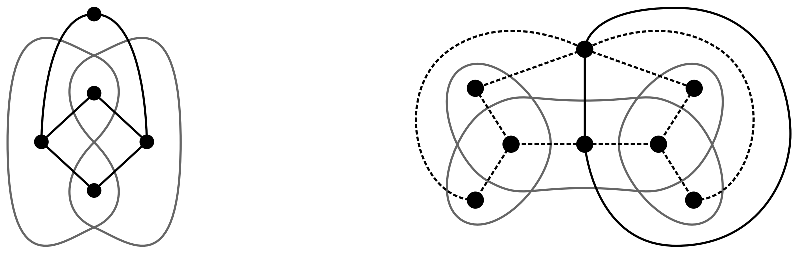


FIG. 4: An example of a non-simple quadrangulation (right) and its dual graph (left).

This is not the same thing as prime and composite links– we can assign crossings and get a prime diagram of a composite link or a composite diagram of a prime link– but the theory is quite similar. The biggest difference, as we will see below, is that while connect sum for knot types is associative, connect sum for knot shadows is not.

Definition 11. Given edges e and e in pd-codes $\mathfrak{L}$ and $\mathfrak{L}'$, we can construct a new pd-code $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$. The edges of $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ are the edges of $\mathfrak{L}$ together with the edges of $\mathfrak{L}'$. The vertices of $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ are the vertices of $\mathfrak{L}$ together with the vertices of $\mathfrak{L}'$ with one change: $+e$ and $+e'$ are swapped ($-e$ and $-e'$ stay in their original positions).

The effect of the definition is to switch the positions of the heads of the edges e and e' in the crossings where they occur. We are not trying to be too specific about notation because the edges will need to be relabeled in the new pd-code (see our code for one possible implementation). We can then prove

Proposition 12. *The pd-code $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ is a genus-0 pd-code where a pair of new faces share the edges e and e'.*

Proof. The new collection of vertices clearly still partitions the new collection of signed edge labels, since the same edge labels occur in vertices as in the original pd-codes $\mathfrak{L}$ and $\mathfrak{L}'$. We are swapping a pair of $+$ labels, so the rule on non-adjacent signs is still obeyed in the new pd-code. This means that we need only check the genus.

If we think of faces of a pd-code as lists of signed edges, the effect of the connect sum operation is to concatenate two pairs of these lists. As Figure 5 shows, the successor of $+e$ in $\mathfrak{L}$ is replaced by the successor of $+e'$ in $\mathfrak{L}'$, and the chain of successors continues as before until returning eventually to the successor of $+e$ in $\mathfrak{L}$. This creates a new face in $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ which is formed by merging two previous faces in $\mathfrak{L}$ and $\mathfrak{L}'$ and contains $+e$ and $+e'$. We can argue similarly that a second new face of $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ is also created, containing $-e$ and $-e'$.

Since number of vertices and edges of $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ is the sum of the vertices and edges in $\mathfrak{L}$ and $\mathfrak{L}'$ and the number of faces is 2 less than the sum of the number of faces in $\mathfrak{L}$ and $\mathfrak{L}'$, $V - E + F = 2$ for $\mathfrak{L}\#_{e,e'}\mathfrak{L}'$ and this is still a genus-0 pd-code, as desired. □

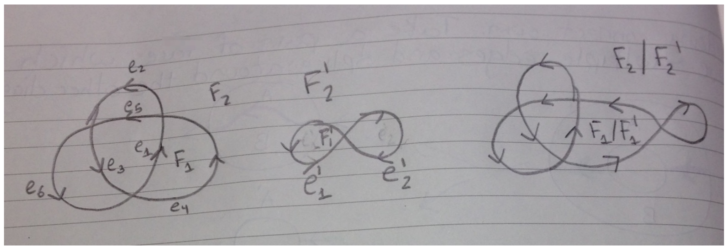


FIG. 5: The connected sum of two link shadows yields a new shadow where two pairs of faces have been merged.

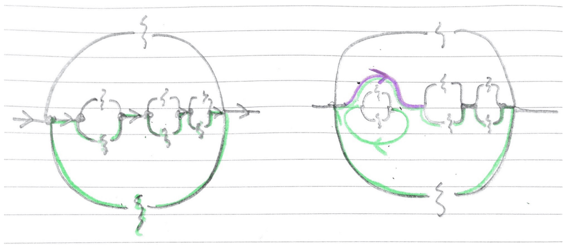


FIG. 6: Two faces which share more than one edge must create an opportunity for a cut and splice.

We now give a theorem corresponding to the prime decomposition of links [9]:

Proposition 13. *Every composite link shadow can be created by connect sum operations on a well-defined set of prime link shadows.*

Proof. We proceed by induction on the number of pairs of edges shared by two faces. If no two faces share more than one edge, the number of pairs is zero and the diagram is already prime. This is the base case.

Observe that as in Figure 6, if some pair of faces shares some number of edges, the shared edges occur in the same order on both faces (if not, the diagram isn't planar). This means that our diagram is the connect sum of two subdiagrams created by cutting and splicing a pair of adjacent shared edges, as on the right side of that Figure.

Since each subdiagram has fewer pairs of edges shared by two faces, by induction we may write them as connect sums of collections of prime diagrams. But we made arbitrary choices when deciding which pair of adjacent edges to cut-and-splice, and so must check that the collection of prime diagrams did not depend on these choices.

It's enough to show that the collection of subdiagrams produced by doing any two such cut-and-splices in one order is the same as the collection of subdiagrams produced by doing them in the other. But this is clear as the operations don't interfere with one another (Figure 7 illustrates the point.) □

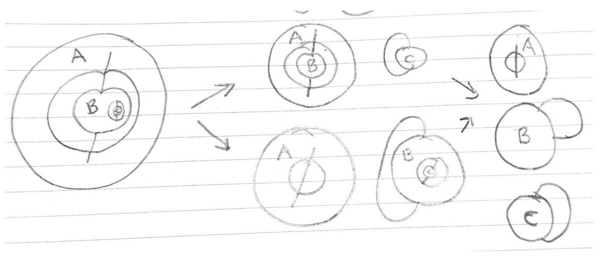


FIG. 7: Cutting and splicing at two locations can be done in either order and produce the same list of subdiagrams.

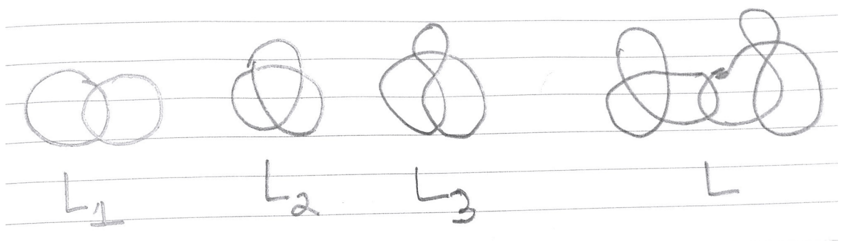


FIG. 8: L can be written as $L = (L_1 \# L_2) \# L_3$ but not as $L_1 \# (L_2 \# L_3)$.

We now know that every n vertex composite shadow L is the connect sum of prime shadows $L_1, \ldots, L_k$ and that the numbers of vertices $V_1, \ldots, V_k$ form an integer partition of n. Therefore, we need to enumerate "all connect sums of all link shadows with numbers of vertices that partition n". This is not as trivial as it sounds. Writing the connect sum as $L_1 \# \ldots \# L_k$ is dangerously misleading– though connect sum is associative (and even commutative) on isotopy classes of knots, the same is not true for links and for diagrams. Figure 8 gives an example of a link shadow L is a connect sum of diagrams L_1, L_2 and L_3, which *can* be written $(L_1 \# L_2) \# L_3$ but *can't* be written $L_1 \# (L_2 \# L_3)$. It is easy to see the following.

Lemma 14. *The set of prime summands $\{L_1, \ldots, L_k\}$ of a composite link shadow L can be renumbered so that $L = (((L_1 \# L_2) \# L_3) \# \cdots L_k)$ and L_1 has the largest number of vertices among all the $L_1, \ldots, L_k$. The numbers of vertices $V_2, \ldots, V_k$ in the other prime summands are* not *guaranteed to be in sorted order.*

Proof. Since the overall shadow is connected, we can start connect-summing at any prime summand and build the rest of the shadow from there by undoing the cut-and-splice operations of Proposition 13. Therefore, we are free to choose L_1 to have a maximal number of vertices. $\square$

Lemma 15. *The set of (diagram-isomorphism classes) $\{L_1, \ldots, L_k\}$ of the prime summands of L is a shadow-isomorphism invariant of L. In particular, the set of numbers of vertices $\{V_1, \ldots, V_k\}$ is a shadow isomorphism invariant of L.*

Proof. This follows directly from the fact that the set $\{L_1, \ldots, L_k\}$ is well-defined. □

Now recall that a pdstor (Definition 9) is an ordered collection of canonical pd-codes which represent pd-isomorphism classes. We can define connect sum for pdstors by saying that $P_1 \# P_2$ is the pdstor constructed by adding all $\mathfrak{L}_1 \#_{e1,e2} \mathfrak{L}_2$ where e_i is an edge of some $\mathfrak{L}_i \in P_i$. We now generate composite link shadows iteratively as follows:

```
procedure BuildCompositeShadows($n$)                ▷ Build all link shadows with $\leq n$ vertices
  Use *plantri* to build pdstors $P_1, \ldots, P_n$ containing all oriented prime link shadows with 1 to $n$ vertices.
  for all $1 \leq k \leq n$ do
    Define an empty pdstor $P_k$.
    for all partially sorted partitions $k_1 k_2 \ldots k_\ell$ of $k$ with $k_1 \geq k_i$ do
      Build the pdstor $P_{k_1 k_2 \cdots k_\ell} = ((P_{k_1} \# P_{k_2}) \# P_{k_3}) \# \cdots P_{k_\ell}$.
      (If $((P_{k_1} \# P_{k_2}) \# P_{k_3}) \# \cdots P_{k_{\ell-1}}$ was already computed, we can reuse it.)
    end for
    for all *fully* sorted partitions $k_1 k_2 \ldots k_\ell$ of $k$ with $k_1 \geq k_2 \geq \cdots \geq k_\ell$ do
      Define an empty pdstor $P$.
      for all *partially* sorted partitions $k'_1 k'_2 \cdots k'_\ell$ with the same *set* of $k_i$ do
        Add all elements of $P_{k'_1 k'_2 \cdots k'_\ell}$ to $P$.
        (Some of these will be isomorphic to one another.)
      end for
      Add all elements of $P$ to $P_k$.
      (By Lemma 15, these aren't isomorphic to anything previously computed,
      so we don't need to check for isomorphism with existing elements of $P_k$.)
    end for
  end for
end procedure
```

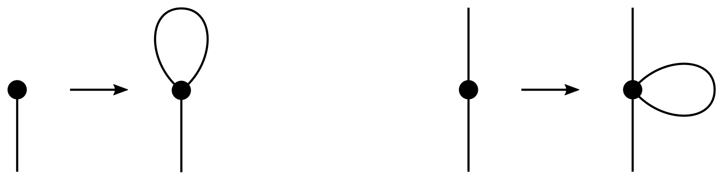
3.2. Expansions of embedded planar simple graphs

Our next strategy will be much more complicated (and somewhat slower to run), but it serves as crucial check on the previous computation. The basic idea is to define a smaller class of graphs so that the graphs we are interested in can be obtained from the base class of graphs by various expansion moves. Lehel [10] gave a strategy for generating all 4-regular graphs in this way from the octahedral graph. Instead of using Lehel's strategy directly, we build on the method of Brinkmann and McKay [2, 12] for enumerating isomorph-free embedded planar graphs; we extend their work here to generate the class of graphs that we're interested in.

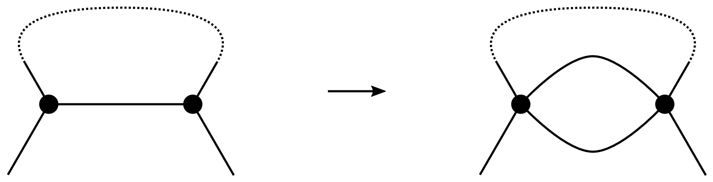
We observed above that the link shadows are embedded isomorphism classes of 4-regular embedded planar multigraphs. We now define four expansion moves of embedded planar graphs with vertex degree ≤ 4 which generate embedded planar multigraphs of vertex degree ≤ 4 with the same number of vertices, but additional edges:

Definition 16. The four expansion operations that we will use are the following:

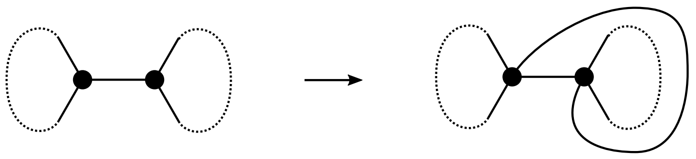
- E_1 Loop insertion adds a loop edge to a vertex of degree 1 or 2, as below. (Note: Loop insertion can be performed on each side of a vertex of degree 2).



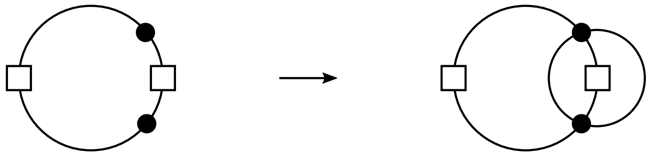
- E_2 reversing edge doubling duplicates an existing edge joining vertices of degree < 4 so as to create a new bigon face. Note that the (counterclockwise) order of the two vertices is reversed on the two vertices.



- E_3 preserving doubling also duplicates an existing edge joining vertices of degree < 4, but keeps the counterclockwise order of the edges the same on each at each of the two vertices. This sort of doubling is only available if the original edge is a cut edge of the graph.



- E_4 pair insertion adds a pair of edges simultaneously, joining two vertices of degree 2 which are both on two faces of the embedding, as below.

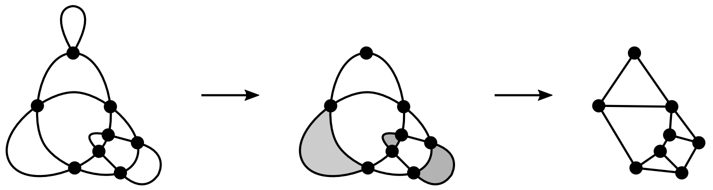


We can now show

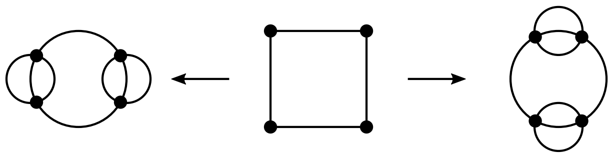
Proposition 17. *Every link shadow L can be obtained from a connected, embedded planar simple graph of vertex degree ≤ 4 G_0 by a series of E_1, E_2, E_3, and E_4 expansions.*

Equivalently, any link shadow L can be reduced to a connected embedded planar simple graph L_0 of vertex degree ≤ 4 by a series of E_1, E_2, E_3, and E_4 reductions. The embedded isomorphism type of L_0 is determined uniquely by the (unoriented) shadow isomorphism type of L (the order in which the reductions are performed doesn't matter).

An illustration of the process we describe is shown below.



The proof appears in Appendix B. We can now build the link shadows by applying expansions to the graphs produced by *plantri*. This is not trivial. First, not all sequences of expansion moves lead to link shadows. Second, different sequences of expansion moves may produce isomorphic link shadows. Some examples are shown in Figure **??**.



We can think of this as the problem of searching for collections of expansion moves which obey various equations, such as the fact that the total vertex degree must be 4 in any complete solution (there are less obvious equations as well– the complete system is specified in Appendix B).

We find all of the solutions using a standard branch-and-bound algorithm and add the results to a pdstor to eliminate isomorphic pd-codes. We can save a lot of time in this process by noting that tells us that we need only check expansions of the same graph (and their reorientations) against each other for potential pd-isomorphisms.

4. COMPUTING AND VERIFYING THE RESULTS

We implemented the above algorithms in C and used them to generate a database of knot diagrams up to 10 crossings. The longest run took several days on a desktop computer. The most

Crossings	Prime Shadows	Link Shadows	Link Diagrams	Knot Shadows	Knot Diagrams
3	1	7	–	6	–
4	2	30	–	19	–
5	3	124	–	76	–
6	9	733	–	376	–
7	18	4586	–	2194	–
8	62	33 373	–	14 614*	–
9	198	259 434*	–	106 421*	–
10	803	2 152 298*	–	823 832*	–

TABLE I: The number of link (including knots) and knot shadows and diagrams through 10 crossings. The unstarred numbers in the left hand column match those of Kápolnai et. al. [**?**], while the unstarred numbers on the right are sequence A008989 in the OEIS. The starred numbers are new.

important question, of course, is how the computation was checked. Of course, our implementation was careful and involved quite a bit of internal self-checks as well as testing against *valgrind* for memory problems, and writing a suite of unit tests for the codebase. However, good programming practices can only provide a limited measure of confidence in the results, so we continued to test our work. The first and most important test was to verify that the databases obtained by connect summing and by expansions were identical.

We were also able to check against some existing enumerations. Kápolnai et. al. classified spherical "multiquadrangulations" [**?**], which are the duals of our diagrams as explained in Section 3.1. Their table 2 of the count of multiquadrangulations matches our count of diagrams exactly through 8 crossings (note that the quadrangulation has two more vertices than the dual knot diagram has crossings, so their data is shifted by two). It's worth observing that these authors also use *plantri*, so their results are not completely independent from ours. Still, it provides some comfort to see that their implementation on top of *plantri* produces the same results as ours. For knot shadows in particular, Arnol'd has given counts of the number of immersions of the unoriented circle into the unoriented sphere with n crossings for n from 0 to 5 [1, page 79]. This is sequence A008989 in the Online Encyclopedia of Integer Sequences, with an extension to $n = 7$ credited to Guy H. Valette. We could not find a published reference for Vallette's extension of the table, but our data for knots does match A008989 including the extension. These counts are given in the table below:

5. CLASSIFYING KNOT TYPES

Our database of diagrams contains all diagrams of knots and links with n crossings. Though very few of these are knot diagrams (the subject of this paper), there seems to be no obvious graph-theoretic way to avoid generating link diagrams. So our next move is to simply compute the number of components in each diagram and throw out the diagrams with more than one component. This gives us the following table of the number of diagrams we've found:

¡data¿

Matt and Harrison write something about assigning crossings and orientations, and give data on expansions

Matt and Harrison write something about their respective knot identification strategies and the verification that they match

6. RESULTS

giant pictures, compared with tait's classification our distributions monogon and bigon fractions degree of alternatingness universal properties? comparison with distribution from ERPs, lattice walks, and petaluma.

Matt will read about polynomials, graph polynomials, and such and try to relate average invariants to graph theoretic properties.

Symmetries are rare!

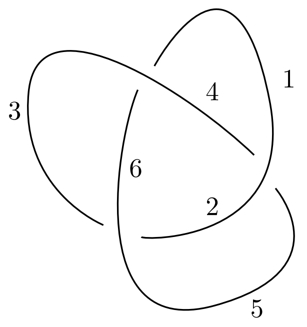
Everything contains any given prime thing as a connect summand! Test unknots, trefoils!

An experiment to do: make a histogram of the number of diagrams which are all unknots, those that are mostly unknots, those that are partially unknots, and so forth.

I think this should partially explain the unknot fraction; the idea is that we should be able to count diagrams which can't possibly be knots (that is, stuff with only R1's) and see that while they eventually become exponentially scarce, they are (in this data) quite prevalent. Of course, it would be better if we also knew an upper bound on the total number of diagrams (maybe from Thistlethwaite's paper?), so that we could get an asymptotic bound on the decay rate.

7. FUTURE DIRECTIONS

More sophisticated counting procedures have been applied in recent years, particularly for diagrams of knots. Jacobsen and Zinn-Justin[**?**] give a transfer matrix analysis for a combinatorial description of prime, reduced, weighted, marked planar diagrams for knots very similar to Gusein-Zade's enumeration [**? ?**]. Gusein-Zade's actual computer enumeration was carried out only to 10 crossings while Jacobsen and Zinn-Justin push the enumeration to the nearly unthinkable 22-crossing case, where they give an exact count of 40558226664529044000 knot diagrams.



$$\{[+4,-2,-5,+1],[+2,-6,-3,+5],$$
$$[+6,-4,-1,+3]\}$$

FIG. 9: A diagram for 3_1 and its PD-code. The labels are only single integers here as there is only one component. Note that we may omit directional arrows as the orientation can be inferred from the ordering of the edge labels.

Schaeffer [**?**] (see also [**?**]) constructed a very insightful bijection between link diagrams where an edge stretches to infinity ("two-leg" diagrams) and a class of decorated trees called "blossom trees". This has been used to great effect in constructing generating functions for the number of these diagrams, and we will use such a generating function due to Bouttier-Di Francesco-Guitter [**?**] to check our work below. Unfortunately, the relationship between our count and theirs is not simple; symmetries of our diagrams confuse the counting procedure so that we must weight our diagrams by a factor derived from the size of their automorphism group to recover the counts of [**?**].

transitions, unknotting number and so forth.

Appendix A: PD-codes and isomorphisms

One of us (Mastin) has given a detailed construction of a combinatorial object, called a PD-code, bijective to the knot diagrams. A PD-code is a set comprised of quadruples, where each entry in a quadruple may be viewed as an edge in a link diagram and each quadruple corresponds to a crossing. PD-codes seem to have been first used by the KnotTheory Mathematica package **cite Knot Atlas** developed by Dror Bar-Natan and his students, but they are very close to the combinatorial description of diagrams used by Millett and Ewing's *lmpoly* code[8].

Definition 18. Given a link diagram with labeled edges we generate the set of quadruples of the **PD-code** representing this diagram by the following procedure. For each crossing we include the quadruple of edge labels involved beginning with the incoming under-edge and proceeding around the crossings in the positively oriented direction of S (see Figure 9). We give a positive sign to incoming edges and a negative sign to outgoing edges.

The details of the bijection between the PD-codes and link diagrams can be found in **cite PD-code Paper**. We will presently include the construction of a cell decomposition of S^2 associated with a PD-code as it will play a central roll in our algorithm for enumerating diagrams. The vertices of the cell decomposition will correspond to the quadruples appearing in the PD-code and the edges will be the edges of the diagram. In order to defined the faces we will introduce the **successor map** which essentially gives the next edge to appear if one were to walk along an edge and turn right a vertex.

Definition 19. If we represent the quadruples in the a PD-code as $[e_i^0, e_i^1, e_i^2, e_i^3]$, then we define the successor map by $s(e_i^j) = -e_i^{j+1 \mod (4)}$.

We can now define a cell complex associated to a PD-code which recovers the link diagram on S^2.

Definition 20. For each orbit of the successor map s we construct an oriented polygon in the plane with sides given by the labels in each orbit where the labels are assigned clockwise around the polygon. The orientation of an edge agrees with the orientation inherited from the standard orientation of the plane for *negative* labels and disagrees with this orientation for *positive* labels. The orientations of the 2-cells is inherited from the standard orientation on the plane. The orientations of the edges is induced by the orientation of the 2-cells. This construction gives the **associated cell complex** to a given PD-code.

This construction gives a collection of oriented polygons in the plane whose edges are labeled by the edge labels of the given link diagram. If these polygons are glued according to the edge labels and orientations then the link diagram is reconstructed as the 1-skeleton of the resulting cell complex. The details of this construction can be found in **cite PD-code Paper**.

Definition 21. Given two PD-codes we say that a bijection between the edge labels is a **PD-code isomorphism** if it induces an isomorphism on the associated cell complexes.

It is clear that a lot of data about a pdcode is preserved by isomorphism: for instance, the number of crossings, edges, faces, and the numbers of edges around faces. We can use this information to rule out isomorphisms using a hashing scheme.

Definition 22. Suppose the pdcode P has V crossings, E edges, F faces, and C components. We assume that the faces are denoted $f_1, \ldots, f_F$ and the components are denoted $c_1, \ldots, c_C$. Further, let $\mathrm{edges}(x)$ give the number of edges on a face or component. Then the *hash* of P is given by the tuple

$$\mathcal{H}(P) = (V, E, F, C, \{\mathrm{edges}(f_1), \ldots, \mathrm{edges}(f_F)\}, \{\mathrm{edges}(c_1), \ldots, \mathrm{edges}(c_C)\}).$$

The last two are unordered sets of integers.

It is clear that

Lemma 23. *If two pdcodes P_1 and P_2 are isomorphic, then $\mathcal{H}(P_1) = \mathcal{H}(P_2)$.*

Proof. The numbers V, E, F, and C are clearly preserved by isomorphism. The indices of edges and faces may be permuted by an isomorphism, but the number of edges on each can't change. Thus the *unordered sets* of edge counts for faces and component remain the same as well. $\square$

We can now build up an isomorphism between pdcodes by a series of definitions:

Definition 24. Suppose we have two pdcodes P and P' with the same hash.

- A bijection $\gamma : \{c_1, \ldots, c_C\} \to \{c'_1, \ldots, c'_C\}$ between the components of P and the components of P' is called *component-length preserving* if $\#\,\text{edges}(c_i) = \#\,\text{edges}(\gamma(c_i))$ for all i.
- Given such a component-length preserving bijection γ, a bijection $\epsilon : \{e_1, \ldots, e_E\} \to \{e'_1, \ldots, e'_E\}$ between the edges of P and the edges of P' is called *component-preserving* and *compatible with* γ if ϵ maps the edges of each c_i to the edges of $\gamma(c_i)$ by an element of the dihedral group $D_{\text{edges}(c_i)}$. That is, the edges of c_i are mapped in cyclic (or reverse-cyclic) order to the corresponding edges of $\gamma(c_i)$.
- Given a component-preserving bijection $\epsilon : \{e_1, \ldots, e_E\} \to \{e'_1, \ldots, e'_E\}$ between the edges of P and the edges of P', we say that a bijection $\nu : \{v_1, \ldots, v_V\} \to \{v'_1, \ldots, v'_V\}$ between the vertices of P and the vertices of P' is *compatible with* ϵ if

  $$\nu(\text{head}(e_i)) = \text{head}(\epsilon(e_i)) \quad \text{and} \quad \nu(\text{tail}(e_i)) = \text{tail}(\epsilon(e_i)$$

  when e_i is part of a component mapped by an orientation-preserving element of the dihedral group, and

  $$\nu(\text{head}(e_i)) = \text{tail}(\epsilon(e_i)) \quad \text{and} \quad \nu(\text{tail}(e_i)) = \text{head}(\epsilon(e_i)$$

  when e_i is part of a component mapped by an orientation-reversing element of the dihedral group.
- Given γ, ϵ, and ν that obey all the above conditions, we say that they are:
  - *globally orientation-preserving* if the set of edges e_i, e_j, e_k, e_l incident to each vertex v of P (in counterclockwise cyclic order) is mapped to the set of edges $\epsilon(e_i), \epsilon(e_j), \epsilon(e_k), \epsilon(e_l)$ incident to $\nu(v)$ in counterclockwise cyclic order.
  - *globally orientation reversing* if the $\epsilon(e_i), \epsilon(e_j), \epsilon(e_k), \epsilon(e_l)$ are incident to $\nu(v)$ but in clockwise cyclic order (for each v),
  - otherwise, the triple is *inconsistent*.

We then have

Proposition 25. *Given a pair of pdcodes P and P' with the same hash, each isomorphism of P to P' is given by a set of bijections γ between their components, ϵ between their edges, and ν between their vertices where*

- *γ is component-length preserving,*
- *ϵ is component-preserving and compatible with γ,*
- *ν is compatible with ϵ*

and the triple is globally orientation preserving or reversing (not inconsistent).

Proof. Basically, this is the definition of pd isomorphic. □

A few other observations are helpful:

Lemma 26. *If $\mathcal{H}(P) = \mathcal{H}(P')$ for pdcodes P and P', there is at least one component-length preserving $\gamma : \{c_1, \ldots, c_C\} \to \{c'_1, \ldots, c'_C\}$. All component-length preserving γ can be generated by iterating over a product of permutation groups.*

Proof. We can partition the components of P (and P') into subsets by component length. Since the hashes $\mathcal{H}(P)$ and $\mathcal{H}(P')$ match, these partitions correspond for P and P'. The component-length preserving γ are exactly those bijections which respect this partition: we can construct all such γ by taking a product of permutation groups whose orders are the size of the subsets. At worst, all subsets have size 1 and there is a single such γ. □

Lemma 27. *Given a component-length preserving γ and component-preserving and compatible ϵ, and a set of orientations for the components of P, there is at most one $\nu : \{v_1, \ldots, v_V\} \to \{v'_1, \ldots, v'_V\}$ which is compatible with ϵ and consistently oriented on components and we can construct ν as below.*

Proof. Each vertex v of P is incident to four edges e_i, e_j, e_k, e_l. Without loss of generality, let's assume that $v = \operatorname{tail}(e_i), \operatorname{tail}(e_j), \operatorname{head}(e_k)$ and $\operatorname{head}(e_l)$. Then if ν is compatible with ϵ, we must have

$$\nu(v) = \operatorname{tail}(\epsilon(e_i)) = \operatorname{tail}(\epsilon(e_j)) = \operatorname{head}(\epsilon(e_k)) = \operatorname{head}(\epsilon(e_l)).$$

If the four terms on the right are equal, this uniquely defines $\nu(v)$. If not, there is no compatible ν. □

```
procedure BuildIsomorphisms($P$,$P'$)        ▷ Build isomorphisms between pdcodes $P$ and $P'$
    if the hashes $\mathcal{H}(P)$ and $\mathcal{H}(P')$ are different then
        $P$ and $P'$ are not isomorphic. Return $\emptyset$.
    end if
    for all component-length preserving $\gamma : \{c_1, \dots, c_C\} \to \{c'_1, \dots, c'_C\}$ do
        for all compatible and component-preserving $\epsilon$ do
            if a compatible $\nu$ exists then
                if $\nu$ is globally orientation preserving or reversing then
                    $\epsilon$, $\nu$ define an isomorphism $P \to P'$
                end if
            end if
        end for
    end for
end procedure
```

We can now find all isomorphisms between two pdcodes computationally by a simple brute-force strategy:

It should be immediately clear that there are smarter ways to organize this computation: for instance, we could build only part of γ and ϵ before we start checking the consistency conditions for the existence of a compatible ν. Or we could map faces to faces by elements of the dihedral group on the grounds that faces are smaller than components. Or we could try for a more complicated hashing scheme which took into account the adjacency relations between faces. These sorts of ideas are implemented with great insight and care in graph isomorphism packages like *nauty* and *saucy*. However, making these types of changes makes the code more intricate and difficult to debug: in this project, our overall strategy is to use simple, straightforward algorithms in the hopes of minimizing the chance of errors.

Appendix A: PD-codes and isomorphisms

Proof. We will prove the second statement, reducing in stages from some $G_n = G$ to G_0 by performing one reduction at each step. The number of steps we can perform is clearly finite, since each reduces the number of edges by at least one. So suppose we are at stage G_i. If there are no loop or multiple edges, we're done, and this is the simple graph G_0.

If there is a loop edge, we can remove it with a E_1 move.

If there is a multiple edge, we must consider several cases. We can think of each vertex of G_i as retaining a list of 4 connection points, ordered counterclockwise, from the initial embedding of G. Since we have performed some reductions already, some of these may be empty, but at least two are filled at each end of the multiple edge. Pick one vertex of the multiple edge and call it v and the other vertex w.

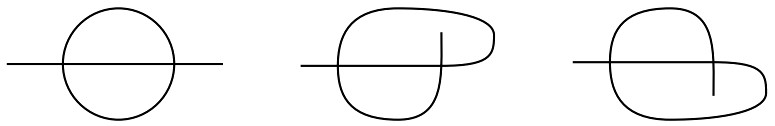
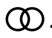


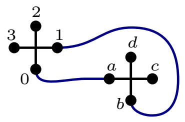
FIG. 10: By parity, because we came from a 4-regular embedded planar graph only the leftmost case can occur at any stage in the reduction process.

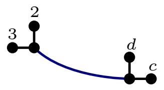
If the edge multiplicity is four, G is . This is obtained from the graph with one edge and two vertices by three E_2 moves.

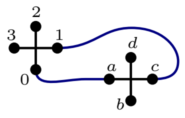
If the edge multiplicity is three or two, there is at least one connection point on v which is not occupied by a copy of the multiple edge followed immediately by a connection point which is occupied by a copy e of the multiple edge. Without loss of generality, we'll call e the *base copy* of the multiple edge, and its connection point to at v position 0 around v. The remaining connection points will be numbered 1, 2, and 3. By construction, the edge joined to v at position 3 (if any) is not connected to w. We can label the other end of the base copy e position a on the second vertex w, and label the other positions b, c, and d, again counterclockwise.

If the edge multiplicity is three, only one of these positions is unoccupied by a copy of the multiple edge. Looking at the three cases (shown in Figure 10), we can see that by parity, it must be position b, and the pair of copies $0a$ and $2c$ of the multiple edge can be removed by a E_4 operation. We have now disposed of the case where edge multiplicity is three.

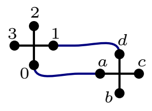
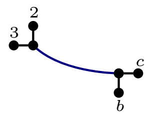
If edge multiplicity is two, there is one edge unaccounted for, which joins either position 1 or 2 on vertex v to position b, c, or d on vertex w. Therefore, there are six cases to address. We consider them in order, starting with the $1x$ configurations.

- 

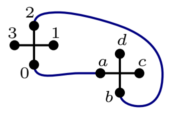
In the $1b$ configuration, the multiple edge forms a 2-cycle dividing the portion of the graph G connected to cd from the portion connected to 23. Deleting $1b$ requires a E_3 move, and the remaining base edge is a cut edge of all further-reduced G_i, as shown at right.


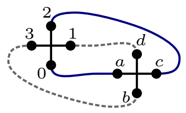
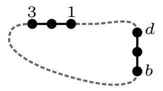
- 

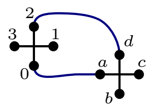
The $1c$ configuration is forbidden by parity.

- 
In the $1d$ configuration, the multiple edge forms a bigon face. Deleting $1d$ uses an E_2 reduction, and yields the configuration at right. The remaining base edge may or may not be a cut edge of the further G_i.


One might think that the $2-$ configurations are simply rearrangements of those above, but this is not true. A genuinely new case arises for $2c$.

- 
The $2b$ configuration is forbidden by parity.

- 
In the $2c$ configuration, by parity, the graph G must have connected 1 and d and also 3 and b. None of our moves change the connectivity of the graph (because we never delete all copies of a multiple edge), so the current graph G_i still joins these pairs of connection points. This means that we are in position for an E_4 pair reduction, resulting in the graph at right.


- 
The $2d$ configuration is forbidden by parity.

Along the way, our analysis has been entirely local: we need only consider a single vertex to decide whether we can apply an E_1 reduction and a pair of vertices to decide on E_2, E_3, and E_4 operations. To show that order of operations doesn't matter, we need to show that whether or not we can apply these operations does not depend on which reductions have already been performed. First, we note that since we never remove all copies of multiple edge, we never change the connectivity of the graph during the reduction process.

The three copies of a multiplicity three edge must bound two bigons, and this does not change as we reduce other edges. Therefore, the E_4 move is always available for all multiplicity three edges.

Whether a multiplicity two edge is eligible for an E_2 move depends only on the positions of the ends of the multiple copies on their vertices, which doesn't change as we reduce. Therefore, this operation can always be performed (or is always forbidden), regardless of which reductions have already been performed.

Whether a multiplicity two edge is eligible for a E_3 or E_4 operation depends not only on the positions of ends of edges on their vertices, but also on the connectivity of the (reduced) graph.

However, as we noted above, the connectivity of the graph doesn't change as we perform reductions.

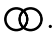
It is clear that the isomorphism type of G_0 does not depend on the order of reduction– after all, in the end we are simply reducing the multiplicity of multiple edges of the edge.

It takes only a moment longer to realize that the embedding of G_0 is determined as well– this embedding is determined by the cyclic order of (surviving) edges around their vertices. We will have deleted some edges from many of these vertices by the time we reach G_0, potentially leaving many empty connections. However, the cyclic order of the surviving edges won't be affected by the order in which these connections were emptied.

One might worry that the choice of *which*[1] copy of an edge of multiplicity two to delete could affect the embedded isomorphism type after an E_2 or E_3 reduction, but it's easy to check that the two possible reduced configurations are (embedded) graph isomorphic by looking at the pictures above. Formally, the point is that the two copies of the edge are adjacent in the cyclic ordering of edges at each vertex, so the surviving copy is always in the same cyclic position relative to surviving edges incident to the vertex. $\square$

We can use this theorem to come up with a strategy for generating diagrams. Basically, we will start by enumerating embedded planar simple graphs of vertex degree ≤ 4 using *plantri* , then expand them to 4-regular embedded planar graphs using the moves above. Afterwards, we will see that we can generate embedded isomorphic graphs with different expansion sequences, so we will have to filter the graphs into isomorphism classes. We start with two lemmas:

Lemma 28. *If the embedded planar graph of vertex degree ≤ 4 G_0 is obtained from a 4-regular embedded planar multigraph G by the reduction process of Proposition 3.2 then either every vertex of degree one in G_0 has exactly one loop edge in G and one multiedge of multiplicity two obtained by E_2 or E_3 or the graph is* .

Proof. If we expand G_0 to G using the four moves, three empty connections on the vertex must be filled during the process. If they are filled by redoubling the existing edge, then the degree of the vertex at the other end of the edge was also one, and we get . Otherwise, we must fill two by adding a loop edge, and the other by doubling the existing edge. $\square$

Lemma 29. *Two pairs of vertices ab and cd on the unit circle may be joined by nonintersecting chords inside the circle if and only if the pairs are unlinked on the circle. That is, if ab and cd are adjacent in the cyclic ordering of the four vertices, as opposed to an order such as $acbd$ or $adbc$ where the pairs alternate.*

[1] Remember that the choice of "base edge" was arbitrary.

We can now design an algorithm to produce all possible expansions of G_0, a given connected embedded planar simple graph of vertex degree ≤ 4 as an integer constraint satisfaction problem. By Lemma 28, we must add a loop to each vertex of degree one in G_0 eventually. We can save time by doing so at the start of the computation. We will therefore assume that loops have been added to create a *prepared* graph G_1, and each vertex has degree 2, 3, or 4.

We will now define four classes of variables:

- l_i for every vertex v_i of degree 2
- $d_{i,j}$ for every non-cut edge e_{ij} in the graph joining vertices of degree < 4.
- $c_{i,j}$ for every cut edge e_{ij} joining vertices of degree < 4.
- $p_{i,j}$ for every pair of vertices v_i, v_j which both have degree 2 and are both on two different faces of the embedding

We take the subscripts to be unordered. That is, $d_{4,17}$ and $d_{17,4}$ are the same variable, since the edges $e_{4,17}$ and $e_{17,4}$ are the same edge.

These variables will all be valued in $0-1$, and represent the absence or presence of E_1 loop edges, E_2 or E_3 doubles of existing edges and E_4 insertions of new pairs of edges. We can now define two sets of equations relating these variables.

Definition 30. We define the *vertex degree equations* for a prepared graph G_1 to be the collection of equations indexed by the vertices of G_1 given below. For each vertex index i of degree $\delta(i)$ in G_1

$$\delta(i) + 2l_i + \sum_j d_{i,j} + \sum_j c_{i,j} + 2\sum_j p_{i,j} = 4$$

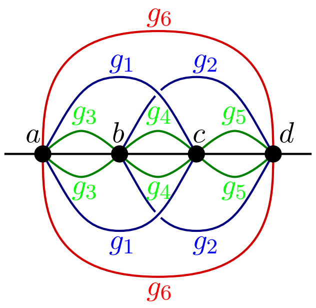
where the sums are taken over all j for which the appropriate variables exist. These equations express the fact that in a complete expansion, the vertex degrees must all be four.

The pair variables $p_{i,j}$ satisfy an additional set of equations:

Definition 31. For each $p_{i,j}$ and $p_{k,l}$ so that the vertices v_i, v_j, v_k and v_l are on the same pairs of faces, and so that the vertices are in the (cyclic) order v_i, v_k, v_j, v_l or v_i, v_l, v_j, v_k we have an additional *linking equation*

$$p_{i,j} + p_{k,l} \leq 1$$

These equations express the fact that the edges corresponding to a linked pair of endpoints along a face must intersect inside the face. Therefore, if two pair variables are linked, at most one of them can take the value 1. For instance, in the situation below where there are four vertices of degree two along a pair of faces, we have six pair variables, two of which obey an additional linking equation.



We note that in the end, at most two of the pair variables above can have value 1, but that a number of combinations are ruled out by vertex degree equations instead of linking equations.

We have defined everything so that

Proposition 32. *Every assignment of $\{0, 1\}$ to the variables l_i, $d_{i,j}$, $c_{i,j}$, and $p_{i,j}$ which obeys the vertex degree equations and linking equations corresponds to an expansion of the connected planar graph G_1 with vertex degrees 2, 3, and 4 and loop edges only to a collection of embeddings for the connected planar 4-regular multigraph G.*

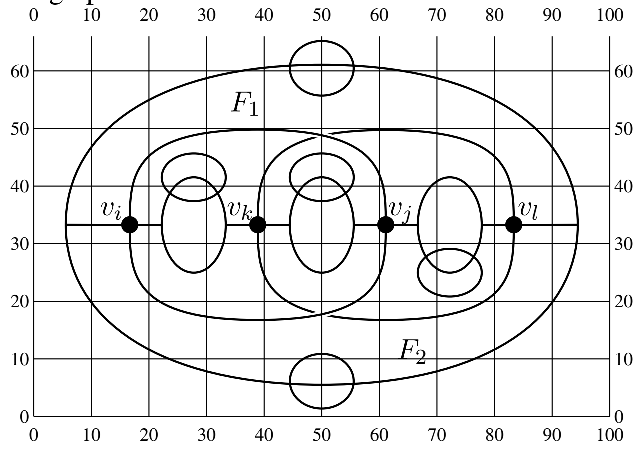
Proof. Actually, there is only a little to check. By the arguments in the proof of Proposition 3.2, the order of expansion moves is irrelevant. So suppose there are n expansions, and we've chosen an order for them, and are trying to generate a family of graphs $G_1, G_2, \ldots, G_n = G$. If we can perform the indicated expansions at all, we will generate a unique connected 4-regular planar multigraph G (we will see that the embedding of G depends on choices we make along the way). So suppose we have generated a given (embedded) G_i, and are trying to expand to G_{i+1}.

If the next expansion is a E_1 expansions indicated by a positive l_i, it is possible as long as the vertex degree at v_i is small enough. This is true, because of the corresponding vertex degree equation. We must choose which side of the edge to insert the loop; each choice yields a different embedding of G_{i+1}, and following the various possibilities will lead to a family of embeddings for $G_n = G$.

If the next expansion is an E_2 indicated by a positive $d_{i,j}$, it is possible as long as the vertex degrees of v_i and v_j are small enough. This is true by their vertex degree equations. There is only one way to make this expansion, leading to a unique embedding for G_{i+1}.

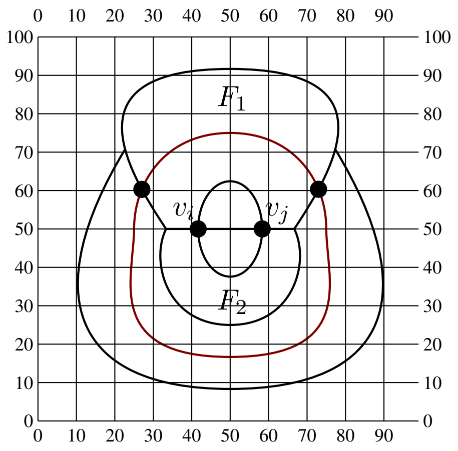
If the next expansion is an E_2 or E_3 expansion indicated by a positive $c_{i,j}$ variable, it is (again) possible if the vertex degrees at v_i and v_j are small enough (which is again true by the vertex degree equations) *and* if $e_{i,j}$ is a cut edge of G_i. We never apply these expansions more than once to an edge, so $e_{i,j}$ is a cut edge of G_i since it was a cut edge of G_1. Choosing between E_2 and E_3 expansions will yield different embeddings of G_{i+1} and we must follow both possibilities to generate the final family of embeddings of G.

This much was easy. If the next expansion is of type E_4, there is more to check. First, we note that there is no ambiguity in embeddings here: if we can do the E_4 expansion, we can do it in only one way and we generate a unique embedding of the graph G_{i+1}. But can we do it at all? Each E_4 indicated by a positive $p_{i,j}$ requires several conditions. First, vertex degrees at v_i, v_j must be small enough, which is checked as before by vertex degree equations. But the v_i and v_j must still be on two faces in the expansion G_i, which is not obvious, because previous E_4 expansions have split faces of G_1 into smaller faces in G_i. Let us suppose that v_i and v_j were on the pair of faces F_1 and F_2 of the original graph.



The pair of faces can make contact with each in several disconnected arcs, as shown above. Further, additional pair edges can share F_1 or F_2 with some other face. However, slicing F_1 can only separate v_i and v_j if the endpoints of the splitting arc link v_i and v_j on F_1. This can't happen if the splitting arc is part of a pair which share F_1 with some other face (as shown), as the interfaces of F_1 and other faces are all connected.

But if the splitting arc also shared F_1 and F_2, the corresponding pair variable $p_{k,l}$ is related to $p_{i,j}$ by a linking equation if and only if adding that arc would leave v_i and v_j on different faces. The linking equation implies that only one of the arcs indicated by $p_{i,j}$ and $p_{k,l}$ is present in the expansion; since we have assumed that $p_{i,j}$ is positive, no such $p_{k,l}$ can have already been inserted earlier in the expansion process. This concludes the case where the interface of F_1 and F_2 was disconnected.

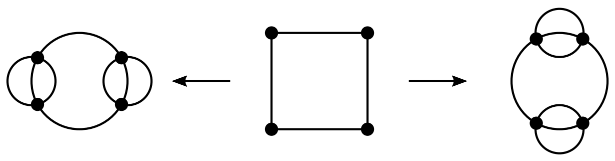


If the interface between F_1 and F_2 is connected, than either might have a disconnected interface with (at most one) other face, as shown above. This case is only cosmetically different– again the key point is that the pair v_i, v_j can link along the boundary of F_1 (or F_2) with a pair edge which also shares F_1 and F_2 while pairs involving a third face won't link the vertices we're interested in.

We last have only to observe that by the vertex degree equations, the final graph G is a 4-regular planar multigraph. Since we have only added edges along the way, G is connected because G_1 was. $\square$

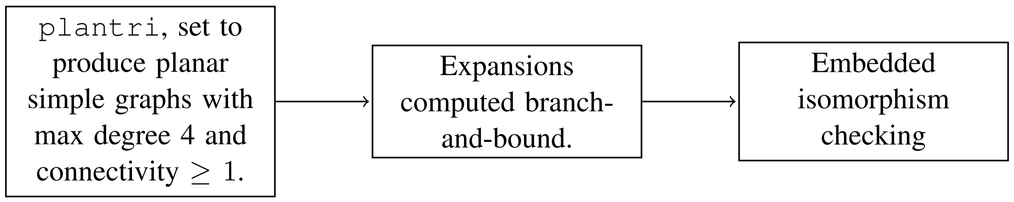
We have reduced the problem to that of building and satisfying the vertex degree and linking equations. This problem is basically standard, and we use the usual branch-and-bound algorithm. We must define a canonical order on the variables (it doesn't matter how, but to be specific, in our implementation we sort the classes of variables in the order $l_i \prec d_{i,j} \prec c_{i,j} \prec p_{ij}$ and in dictionary order by the (sorted) pair $\{i, j\}$ within each class). Then we enumerate the possible assignments of $\{0, 1\}$ to variables recursively, pruning the tree whenever a vertex degree or linking equation is violated. As usual, this is in theory possibly exponentially slow, but in practice efficient enough for small n.

We now consider the problem of dividing the results into embedded isomorphism classes. We first observe that we have already shown in Proposition 3.2 two different reduced graphs G_0 and G_0' cannot expand to the same G since the embedded isomorphism type of the reduction G_0 is determined by the embedded isomorphism type of the expansion. However, it is possible for two different collections of expansion moves for the *same* graph G_0 to produce isomorphic G and G' as in the picture below:



Therefore, we must insert each expansion we generate from a solution to the vertex degree and linking constraints into a container which rejects the solution if an embedded isomorphic graph already exists in the container. Though very fast graph isomorphism checkers such as *nauty* and *saucy* might speed things up, the number of vertices here is very small and we get entirely acceptable performance simply by using a hashing scheme and then attempting to build isomorphisms by pruned search.

The overall workflow is then as follows:


`plantri`, set to produce planar simple graphs with max degree 4 and connectivity ≥ 1.
Expansions computed branch-and-bound.
Embedded isomorphism checking

[1] V I Arnold. Plane curves, their invariants, perestroikas and classifications. *Singularities and Bifwrcations*, 1994.

[2] G Brinkmann, G Brinkmann, B D McKay, and B D McKay. Fast generation of planar graphs. *MATCH Commun. Math. Comput. Chem*, 58(2):323–357, 2007.

[3] G Brinkmann, S Greenberg, C Greenhill, B D McKay, R Thomas, and P Wollan. Generation of simple quadrangulations of the sphere. *Discrete mathematics*, 305(1):33–54, 2005.

[4] Moshe Cohen and Sunder Ram Krishnan. Random knots using Chebyshev billiard table diagrams. *arXiv.org*, May 2015.

[5] Y Diao, C Ernst, and U Ziegler. *Generating large random link diagrams*. Physical and Numerical Models in Knot Theory …, 2005.

[6] Nathan Dunfield, A Hirani, M Obeidin, A Ehrenberg, S Bhattacharyya, and D Lei. Random Knots: A preliminary report, 2014.

[7] Chaim Even-Zohar, Joel Hass, Nati Linial, and Tahl Nowik. Invariants of Random Knots and Links. *arXiv.org*, November 2014.

[8] Bruce Ewing and Kenneth C Millett. Computational algorithms and the complexity of link polynomials. Paris, 1997.

[9] Yoko Hashizume. On the uniqueness of the decomposition of a link. *Osaka Math. J.*, 10:283–300, 1958.

[10] Jenö Lehel. Generating all 4-regular planar graphs from the graph of the octahedron. *Journal of Graph Theory*, 5(4):423–426, 1981.

[11] Xiao-Song Lin and Zhenghan Wang. Integral geometry of plane curves and knot invariants. *arXiv.org*, November 1994.

[12] Brendan D McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26(2), February 1998.

[13] S K Nechaev, A Yu Grosberg, and A M Vershik. Random walks on braid groups: Brownian bridges, complexity and statistics. *Journal of Physics A: Mathematical and General*, 29(10):2411–2433, 1996.

[14] Enzo Orlandini and Stuart G Whittington. Statistical topology of closed curves: Some applications in polymer physics. *Reviews of Modern Physics*, 79(2):611–642, 2007.

[15] Gilles Schaeffer and Paul Zinn-Justin. On the asymptotic number of plane curves and alternating knots. *Experiment. Math.*, 13(4):483–493, 2004.