The Next 350 Million Knots

Benjamin A. Burton
The University of Queensland, Brisbane, Australia

Abstract

The tabulation of all prime knots up to a given number of crossings was one of the founding problems of knot theory in the 1800s, and continues to be of interest today. Here we extend the tables from 16 to 19 crossings, with a total of 352 152 252 distinct non-trivial prime knots.

The tabulation has two major stages: (1) a combinatorial enumeration stage, which involves generating a provably sufficient set of candidate knot diagrams; and (2) a computational topology stage, which involves identifying and removing duplicate knots, and certifying that all knots that remain are topologically distinct. In this paper we describe the many different algorithmic components in this process, which draw on graph theory, hyperbolic geometry, knot polynomials, normal surface theory, and computational algebra. We also discuss the algorithm engineering challenges in solving difficult topological problems systematically and reliably on hundreds of millions of inputs, despite the fact that no reliably fast algorithms for these problems are known.

2012 ACM Subject Classification Mathematics of computing → Topology

Keywords and phrases Computational topology, knots, 3-manifolds, implementation

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.25

Supplementary Material `https://regina-normal.github.io/data.html`

Funding Supported by the Australian Research Council, Discovery Project DP150104108.

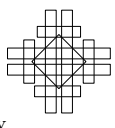
1 Introduction

Knot tabulation is one of the oldest problems in knot theory, dating back to the 1800s when it was thought to be fundamental to the structure of atoms [1]. The last major tabulation work in the literature dates back to 1998, where Hoste, Thistlethwaite and Weeks enumerate all 1 701 936 prime knots with ≤ 16 crossings [23]. In this paper we make the next major leap, tabulating all 352 152 252 topologically distinct non-trivial prime knots with ≤ 19 crossings.

The history of knot theory dates back to Gauss [1], but the first serious tabulation was done by Tait, Kirkman and Little from 1876–1899, culminating in Little's tables for ≤ 10 crossings [25]. Conway extended the tables to 11 crossings in the 1960s [10], and in 1974 Perko unearthed a duplicate 10-crossing pair that had until then gone unnoticed. Dowker and Thistlethwaite extended them to 13 crossings in the 1980s [14], followed by Hoste, Thistlethwaite and Weeks' most recent 16-crossing tables in 1998.

Our goal then is to build a *census* of all non-trivial prime knots that can be drawn with ≤ 19 crossings. Each knot should appear in the census exactly once, up to topological equivalence and/or reflection, using a diagram that uses the fewest crossings possible.

We build our census in two stages. In the first stage, described in section 3, we enumerate a set of *candidate diagrams* that are guaranteed to include every knot in the census, but which may also include unwanted (non-prime) knots and/or duplicates (where the same topological knot appears with different diagrams). In the second stage, described in sections 4–6, we remove duplicate and unwanted knots from our set of candidates, and certify that all remaining knots are prime and topologically distinct.

36th International Symposium on Computational Geometry (SoCG 2020).
Editors: Sergio Cabello and Danny Z. Chen; Article No. 25; pp. 25:1–25:17


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Figure 1 Examples of knot projections.

Computationally, the first stage is enormously easier than the second, both in human effort and computational resources. The first stage uses only combinatorial tools, and took just a few days on a single machine. The second stage involves combinatorics, hyperbolic geometry, knot polynomials, normal surface theory, and computational algebra, and took several months (walltime) with the assistance of a cluster of a few hundred machines.

This project has motivation beyond the "popular science" aspect of knot tabulation (which of course is important for different reasons). An immediate use is to give topologists access to richer data sets for experimentation and exploration. More broadly, computational low-dimensional topology has made enormous progress in recent decades – despite a landscape of algorithmic problems that often range from exponential to tower-of-exponentially slow, are often enormously complex, sometimes unimplementable and occasionally undecidable, practitioners are nevertheless equipping themselves with diverse and sophisticated software tools that can solve these problems effectively in practice. This project showcases how far we have come: we are able to systematically solve complex and difficult problems with at some stages *billions* of inputs, with not one case left unresolved. In a sense, this project offers a blueprint for what large-scale topological computation can look like going into the future.

The bulk of the code is implemented as part of *Regina* [4], and can be downloaded from the master branch of the git repository [7]. We also make significant use of other software, notably *SnapPy* for hyperbolic geometry [13], *GAP* for computational algebra [18], and *plantri* for graph enumeration [3], and we note in sections 3–6 where these tools are used.

The final census includes 352 152 252 knots, including 352 151 858 hyperbolic knots, 380 satellite knots, and 14 torus knots. The full tables, including a more detailed statistical breakdown, can be downloaded from `https://regina-normal.github.io/data.html`.

The computations described here drawn on many diverse branches of mathematics, which we cannot hope to describe properly in this paper. Instead we try to give the flavour of each technique as it is used, and offer the reader references for further reading.

2 Preliminaries

A *knot* is a piecewise-linear simple closed curve in $\mathbb{R}^3$, formed from finitely many line segments. Two knots are *topologically equivalent* if one can be continuously deformed into the other in $\mathbb{R}^3$ without introducing self-intersections. A *non-trivial* knot is one that cannot be deformed into a simple planar polygon.

We typically represent a knot using a *diagram*: a projection of the knot onto the plane with only finitely many double points called *crossings* at which two "strands" of the projection cross transversely, and with no other multiple points at all. See Figure 1 for examples.

The *composition* of knots K and L is formed by cutting them open and connecting them with a bridge, as shown in Figure 2; a *prime* knot is one that cannot be expressed as the composition of two non-trivial knots.

Any two diagrams that represent equivalent knots can be connected through a sequence of *Reidemeister moves*: R1 (twist/untwist); R2 (overlap or pull apart two strands); and R3 (move a strand over a crossing). See Figure 3 for illustrations.

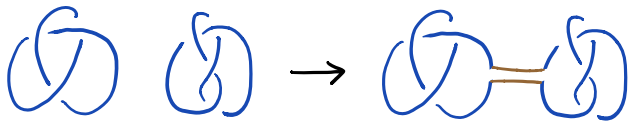


Figure 2 The composition of two knots.

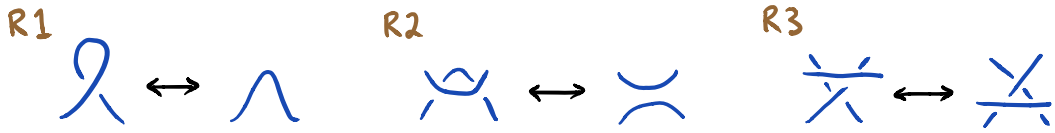


Figure 3 The three Reidemeister moves.

Knots have a close relationship with 3-manifolds. If we extend $\mathbb{R}^3$ with a point at infinity to form the 3-sphere S^3, then the *complement* of a knot K is the 3-manifold $\overline{K}$ formed by drilling out a small regular neighbourhood of K from S^3. Two knots are equivalent if and only if their complements are topologically equivalent; that is, homeomorphic [19].

Thurston's work [31] shows that every knot is exactly one of a *hyperbolic knot*, a *torus knot*, or a *satellite*. Hyperbolic knots have complements that admit a complete hyperbolic metric; torus knots are drawn on an unknotted torus (Figure 4, left); and satellites essentially embed one non-trivial knot inside the neighbourhood of another (Figure 4, right).

3 Stage one: Enumeration

The first stage of building our census is to enumerate a set of candidate diagrams, guaranteed to include each knot from our census at least once.

We do this by first building a candidate set of *model graphs* where each crossing becomes a vertex of degree 4. We reduce this set of graphs using a notion of *flype equivalence*, and then for each graph on n vertices we (essentially) resolve the vertices into crossings in each of the 2^n possible ways. The details follow in sections 3.1–3.3 below.

3.1 Enumeration of model graphs

Here we enumerate our initial set of candidate model graphs. These are all planar 4-regular graphs, and we enumerate not just the graphs but also their possible planar embeddings.

Our first observation is that embedding a graph in the plane is equivalent to embedding the graph in the *sphere* and then selecting one of the resulting cells to be the "outer cell" (i.e.,

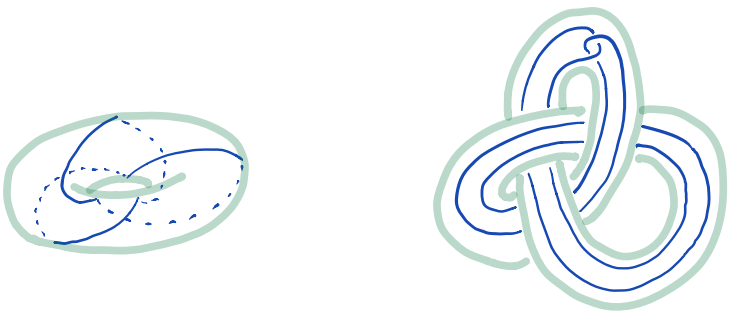


Figure 4 A torus knot and a satellite knot.

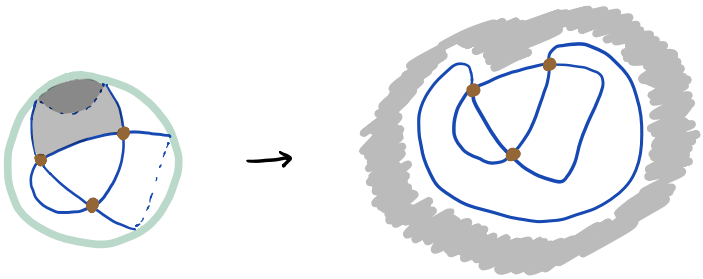


Figure 5 Converting a spherical embedding to a planar embedding.

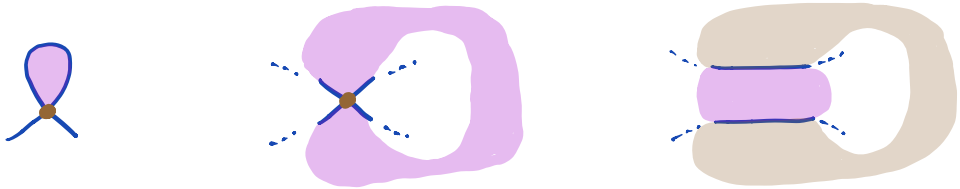


Figure 6 Cells that are not allowed for embeddings of model graphs.

the exterior of the planar embedding); see Figure 5. For our census, it does not matter which cell on the sphere becomes the outer cell: choosing a different outer cell simply corresponds to rotating the knot through 3-dimensional space. We can therefore enumerate 4-regular graphs and their embeddings in the *sphere*, which gives substantially less output.

We put further conditions on this enumeration:

1. We do not allow a cell with just one edge, since any resulting knot diagram can be simplified with move R1 (Figure 6, left).
2. We do not allow a cell that touches itself along a vertex, since any resulting knot diagram can be "untwisted" to use fewer crossings (Figure 6, centre).
3. We do not allow two cells that touch along multiple edges, since any resulting knot diagram would be a composition of two knots, and so would either be non-prime or could be simplified to fewer crossings (Figure 6, right).
4. We insist that, if we follow a path through the graph by always exiting a vertex through the opposite edge from which we enter, then this path must traverse the entire graph. This ensures that any resulting diagram represents a single connected knot, and not a link with multiple disconnected components.

We perform this graph enumeration using the software *plantri* [3]. Conditions 1–3 are supported natively through *plantri* options `-c2m2`, and for condition 4 we extend *plantri* with a simple filter that is tested before each graph is output.

The resulting set: 823 708 396 graphs with spherical embeddings.

3.2 Flype equivalence

A *flype* is a move that involves twisting a section of a knot diagram. Specifically, we begin with a connected region A of a diagram that has exactly four outgoing strands, and where two of these strands immediately meet to form a crossing. The move involves twisting region A upside-down, so that the original crossing disappears, and instead the other two outgoing strands form a new crossing in its place. See Figure 7 (left) for an illustration. We can define a similar move on a model graph, as shown in Figure 7 (right).

Figure 7 Performing a flype on a knot and a model graph.

Figure 8 Forced resolutions for subgraphs.

Tait first observed that, if two model graphs G, H are related by a flype, then any knot diagram modelled by G can be flyped into a knot diagram modelled by H with exactly the same number of crossings [23]. We can therefore partition our model graphs into equivalence classes where the graphs in each class are related by sequences of flypes, and it is enough to use just one representative from each class to build our census of knots.

Because we are dealing with hundreds of millions of graphs, we implement this using union-find [11]. The trade-off is that we need to keep all model graphs in memory simultaneously: for 19 crossings and heavily optimised data structures this required 41 GB of RAM.

The resulting set: 51 280 976 graphs with spherical embeddings.

3.3 Resolving into knot diagrams

For each vertex v of each model graph G, there are two ways of resolving v into a crossing (according to which strand runs above or below the other). This means that, if the graph G has n vertices, it resolves into 2^n knot diagrams.

This expands the size of our candidate set by many orders of magnitude, and so even at this early stage we implement some simple heuristics to reduce the output size:

- Whenever we have a subgraph σ that is (i) two parallel edges or (ii) two parallel edges that form a triangle with a third crossing, we resolve the subgraph in one of the two ways that does not allow a simplification using move R2 or a twist; see Figure 8. As a result, we use only two of the total (four or eight) possible resolutions of S.
- For each knot diagram K, we perform a short random sequence of R3 moves. If we ever obtain a diagram that can be simplified using R1 or R2, we delete K from our output.

The resulting set: 21 004 314 525 candidate knot diagrams.

4 Stage two: Uniqueness

In the second stage, we need to (i) remove duplicates, i.e., different diagrams that represent the same topological knots; (ii) remove non-prime knots (if any); and (iii) certify that all remaining knots are topologically distinct.

In hindsight, as we start stage two, our candidate set is enormous – roughly 60 times the final size of the census. Even disk space is now a serious problem: with compact representations of diagrams, the candidate set still consumes over 1 TB of space. As a result:

- We interleave tasks (i)–(iii) throughout stage two. This is because computational topology often exhibits a time/power trade-off (e.g., how fast an invariant is to compute versus how well it distinguishes different knots). How we choose to resolve this trade-off must change as our candidate set slowly shrinks but its members grow more resistant to simpler tests. Therfore we must choose the *order* of our individual tests carefully.

- Many of our algorithms work in a single read-write pass through the candidate set. We cannot hold all of our candidates in memory at once (so techniques such as union-find are completely out of reach). Even simple operations such as sorting become extremely expensive, and so must be used sparingly.

Here in section 4 we describe an initial set of operations that we perform on all candidate diagrams. The last of these initial operations splits the candidates into hyperbolic vs non-hyperbolic knots; sections 5 and 6 then outline the very different ways that we handle the remaining candidates in each category.

4.1 Exhaustive simplification via R3 moves

First we partition our candidates into equivalence classes, where the knot diagrams in each class are related by sequences of R3 moves (which do not change the number of crossings).

We do this as follows:

- For each input diagram K, we (i) use a breadth-first search to generate all possible diagrams that can be obtained through R3 moves; and then (ii) replace K with the lexicographically smallest of these diagrams. Essentially, this replaces K with a canonical representative of its equivalence class.

  This is computationally wasteful: if an equivalence class has k members then we generate the entire class k times over. We do it this way because, as described above, more sophisticated techniques such as union-find require too much memory, and this method works in a single read-write pass through the entire data set.

  In our breadth-first search we need a way to identify if a knot diagram has been seen before. For this we use *Regina*'s *knot signatures*: short strings computed in small polynomial time from a knot diagram that are the same for two diagrams if and only if the diagrams are combinatorially isomorphic. Our notion of isomorphism here treats knot diagrams as embedded in the sphere, not in the plane; see section 3.1 for details.
- We then sort the resulting list of diagrams lexicographically. This effectively groups together the members of each equivalence class by writing repeated copies of the same canonical representative, and we finish by stripping out any repeats that we see.

  With very tightly engineered data structures and a specialised large memory machine, we were able to do this sort-and-strip in memory, not on disk. However, just this sort-and-strip required 678 GB of RAM, and took a little over a day to run.

The resulting set: 7 205 537 550 candidate knot diagrams.

4.2 Attempting canonical triangulations

Next we make our first use of tools from hyperbolic geometry (despite not yet knowing which of our knots are hyperbolic). For this we work with the software *SnapPy* [13].

It is known that every hyperbolic 3-manifold with torus boundary has a *canonical cell decomposition* [15]. With appropriate subdivision this can be made into a *canonical triangulation*, and Weeks describes an algorithm for computing this triangulation that, whilst not guaranteed to terminate, works effectively in practice [32].

For each candidate knot K, we: (i) build the complement $\overline{K}$, (ii) ask *SnapPy* to find a complete hyperbolic metric on this 3-manifold $\overline{K}$, and if it is successful then we (iii) ask *SnapPy* to compute the canonical triangulation τ of $\overline{K}$.

Several things can go wrong here. First, *SnapPy* uses numerical algorithms with floating point approximations[1], and so it may incorrectly decide that $\overline{K}$ is hyperbolic and/or it may compute a triangulation τ of $\overline{K}$ that is not canonical. Nevertheless, *SnapPy* computes canonical triangulations by making local moves that preserve the topology [32], and so regardless of whether τ is canonical, it is guaranteed that τ *is* topologically the same as $\overline{K}$. Therefore, regardless of what might go wrong, we are guaranteed that if two inputs produce the same "attempted" canonical triangulation τ then the two knot complements are homeomorphic as 3-manifolds, and therefore the two knots are identical.

So: if several knots produce the same "attempted" canonical triangluation then we keep just one of these knots in our candidate set (in particular, one with the smallest number of crossings). For each knot where *SnapPy* fails to produce a hyperbolic structure or canonical triangulation, we simply keep the knot in our set.

On one triangulation at a time, *SnapPy* is extremely fast in practice; however, its algorithms are non-trivial, and for 7 billion triangulations they are far too slow to run in serial. We therefore split the candidate set into pieces which we process in parallel on a cluster, keeping both the knots and the canonical triangulations; afterwards we merge the results together using a similar sort-and-strip process as was used before.

Another point worth noting is *SnapPy* crashed ocassionally due to numerical instabilities – whilst it is enormously effective software in practice, its rare numerical instabilities become common when run through 7 billion inputs. This required specialised code that intercepted and handled crashes automatically without needing a human to babysit, restart and extract partial results from jobs on the cluster.

The resulting set: 367 000 154 candidate knot diagrams.

4.3 Separating hyperbolic from non-hyperbolic

Our next aim is to separate the hyperbolic knots from the non-hyperbolic (satellite and torus) knots, since these two classes will need very different tools going forwards.

To rigorously certify that knots are hyperbolic, we use *strict angle structures*. These are due to Casson and Rivin [28]; essentially they assign a positive internal dihedral angle to each edge of each tetrahedron of a triangulation so that (i) opposite edges in each tetrahedron have equal angles; (ii) all six angles in each tetrahedron sum to 2π; and (iii) the angles around each edge of the triangulation sum to 2π. The key fact for us is a theorem of Casson that, if an orientable 3-manifold triangulation with torus boundary has a strict angle structure, then the underlying manifold admits a complete hyperbolic metric [17].

So, for each input knot K:

- We ask *SnapPy* to find a complete hyperbolic metric on $\overline{K}$ as before, using its numerical algorithms that do not guarantee correctness. If this fails then we retriangulate $\overline{K}$ using local moves that preserve the topology and try again.
- If, after 40 retriangulations, *SnapPy* still fails to find a hyperbolic structure then we mark K as **potentially non-hyperbolic**. Note that this does not *certify* that K is non-hyperbolic: *SnapPy* might have failed, or we might be working with a "bad" triangulation that does not natively support the hyperbolic metric on $\overline{K}$.

[1] *SnapPy* is able to do verified computations, but these are too fragile to run at such an enormous scale – we return to them in later sections where the candidate set is much, much smaller.

- If *SnapPy* claims to find a hyperbolic structure on a triangulation τ of $\overline{K}$, then we attempt to *certify* that K is hyperbolic by finding a strict angle structure on τ. We do this with *Regina* using linear programming with exact arithmetic, and so the result is certified. If we do find a strict angle structure then we mark K as **certified hyperbolic**.
- If *SnapPy* finds a hyperbolic structure but *Regina* does not find a strict angle structure then we mark the knot K as **unusual**.

The resulting set: 352 160 183 certified hyperbolic knots, 14 839 971 potential non-hyperbolic knots, and no unusual knots.

5 Stage two, continued: Finishing the non-hyperbolic case

Here we finish processing the 14 839 971 potential non-hyperbolic knots. By extrapolating from smaller censuses and/or knowing what torus and satellite knots we expect to see, we would only expect a few hundred non-hyperbolic knots at most. Therefore it seems reasonable to guess that almost all of these 14 million potential non-hyperbolic knots are duplicates, and so we focus our initial efforts on stripping these duplicates out.

5.1 Exhaustive simplification, 1 extra crossing

We begin by exhaustively attempting to simplify each knot (as opposed to the randomised attempt in section 3.3). For each knot diagram K with n crossings, this involves a breadth-first search that explores all possible diagrams that can be obtained through *any* sequence of Reidemeister moves, allowing for $\leq n+1$ crossings at any stage. If we ever reach a diagram with $< n$ crossings then we know that K is not minimal, and so it must be a duplicate of another smaller knot in our set.

This is an expensive operation: the number of diagrams reachable from K is potentially exponential in n. (This is why we only run this process over our 14 million potential non-hyperbolic knots, and not our 352 million hyperbolic knots). As in section 4.1, we use knot signatures to ensure that we do not revisit the same diagram more than once.

The resulting set: 3 326 443 potential non-hyperbolic knots.

5.2 Pass moves

We next attempt simplification moves that are more complex than the standard Reidemeister moves. These are *pass moves*, which have been used with great success in knot tabulation since the late 1800s [23, 25]. A pass move involves taking a strand that passes over k consecutive crossings, and rerouting it (by moving it above the diagram) so that it passes over ℓ consecutive crossings instead; see Figure 9 for an illustration. A pass move could of course involve a similar procedure with a strand that passes *under* k consecutive crossings instead.

We are interested in finding pass moves where $\ell < k$, which reduce the total number of crossings. Our implementation is straightforward: we identify strands that pass over (or under) a maximal number of consecutive crossings, and use the Floyd-Warshall shortest path algorithm [11] to see if there is a route that crosses through fewer cells in the diagram.

The resulting set: 239 950 potential non-hyperbolic knots.

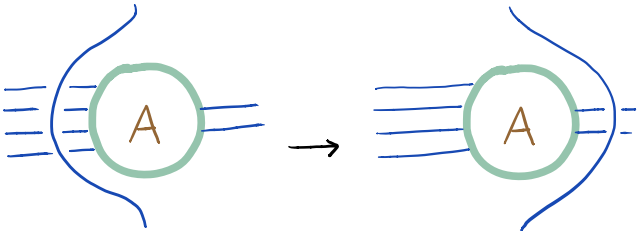


Figure 9 A pass move from $k = 4$ to $\ell = 2$ crossings.

5.3 Exhaustive simplification, 2 extra crossings

We return now to the exhaustive simplification process described in section 5.1, but this time if our input diagram has n crossings then we allow $\leq n + 2$ crossings at any intermediate stage. The number of potential knot diagrams that we can reach increases exponentially with the number of additional crossings that we allow, and so this process is significantly slower than in section 5.1. Nevertheless, we have an order of magnitude fewer knots in our set to begin with, and so the computation remains manageable.

The resulting set: 2 671 potential non-hyperbolic knots.

5.4 Exhaustive search for duplicates, 1 extra crossing

Next we perform another exhaustive search through sequences of Reidemeister moves. However, our aim now is not to simplify one diagram, but rather to find paths that connect different diagrams together.

To do this, for each n, we load all remaining diagrams with n crossings into memory at once, and perform simultaneous breadth-first searches with all of these diagrams as starting points. Whenever two searches connect, we know that the corresponding starting diagrams represent the same topological knot. We continue this process until we have exhausted all reachable diagrams with $\leq n + 1$ crossings.

As with exhaustive simplifiation, the number of diagrams that we could reach is exponential in n; given the large number of starting points, this causes a difficulties with both running time and memory. We are able to relieve the running time somewhat by using a multithreaded implementation of our breadth-first search.

Memory, however, is a serious problem that for now stops us from going beyond $n + 1$ crossings. We resolve this in the next section by splitting the knots into smaller classes.

The resulting set: 2 011 potential non-hyperbolic knots.

5.5 Separating using HOMFLY-PT polynomials

We pause now to split the remaining knots into smaller classes, where it is guaranteed that knots from different classes are distinct. We do this using the HOMFLY-PT polynomial [16, 27], a polynomial invariant that is reasonably strong, has a simple combinatorial interpretation, and can be computed in $O(2^n)$ time (which is reasonably fast in the landscape of knot invariants).[2] In fact the HOMFLY-PT can be computed in sub-exponential time [6], but Kauffman's $O(2^n)$ skein-template algorithm [24] is enough for our purposes.

The resulting set: 2 011 potential non-hyperbolic knots, split into 412 classes.

[2] For each knot K we actually compute both the HOMFLY-PT polynomial of K and the HOMFLY-PT polynomial of the reflection of K, and take whichever is lexicographically smaller. This is necessary because, unlike our census, the HOMFLY-PT polynomial is sensitive to reflection.

5.6 Exhaustive search for duplicates, 2 or 3 extra crossings

We now perform a fresh exhaustive search for duplicates, as in section 5.4, but this time exhausting all reachable diagrams with $\leq n + 3$ crossings. Each extra crossing increases our running time and memory by an order of magnitude; however, we work around this problem by performing a separate exhaustive search only within each HOMFLY-PT class. Since the classes are individually very small, the computation remains manageable.

The results are very pleasing: for *every* one of our HOMFLY-PT classes, we are able to show that *every* diagram is a duplicate of the same knot. This leaves us with just one knot per class, and therefore we have certified that all of our remaining knots are distinct.

The resulting set: 412 potential non-hyperbolic knots, all certified as distinct.

5.7 Certifying hyperbolicity, again

We do not yet know for certain that all 412 of our knots are non-hyperbolic. Since certifying *non*-hyperbolicity is an expensive process, we pass our remaining 412 knots through *SnapPy* once more in case we were unlucky with the random retriangulations the first time around. This follows exactly the same process as in section 4.3, just with different random seeds.

This time we are luckier: the hyperbolic structures and strict angle structures are able to certify 18 of our knots as hyperbolic. The other 394 knots remain potentially non-hyperbolic.

The resulting set: 394 potential non-hyperbolic knots; 18 *new* certified hyperbolic knots.

5.8 Certifying non-hyperbolicity

We now hope that our 394 *potential* non-hyperbolic knots are indeed non-hyperbolic, and so we turn our attention to certifying this.

We begin with the torus knots. These are well-understood: they are completely classified, and it is known exactly how many crossings each torus knot needs. We therefore know in advance that there should be exactly 14 torus knots in our census with ≤ 19 crossings, and we can compute their HOMFLY-PT polynomials. These 14 polynomials indeed appear in our list, and so we know that the corresponding 14 knots are non-hyperbolic.

The remaining 380 knots, if non-hyperbolic, must all be satellites. To certify that a knot K is a satellite, it is enough to find a torus τ embedded in $\overline{K}$ with the property that, if we cut $\overline{K}$ open along τ, the piece containing the boundary of $\overline{K}$ is not the product Torus $\times\, I$, and the other piece is not the solid torus [31].

We use *normal surface theory* to find candidate tori. See [21] for an overview of normal surfaces; the key fact for us is that "important" surfaces in a 3-manifold often appear in the set of *vertex normal surfaces*, which can be constructed using techniques from polytope theory and linear programming. We enumerate this set in *quad-closed coordinates*, which are optimised for working with knot complements; see [9] for details.

For each candidate torus, we use *Regina*'s implementation of solid torus recognition, which – though exponential time – is extremely efficient in practice [8]. To test for Torus $\times\, I$, we fill one of the boundaries with a solid torus in three different ways: by a theorem of Haraway [20], if none of these fillings are solid tori then the manifold is not Torus $\times\, I$.

Although there is no guarantee that the torus we seek *must* appear amongst our candidate set of vertex normal surfaces, this indeed happens for all of our 380 remaining knots.

The resulting set: 394 *certified* non-hyperbolic knots (14 torus knots, 380 satellites).

Figure 10 A sample tangle, and a tangle that includes a knot composition.

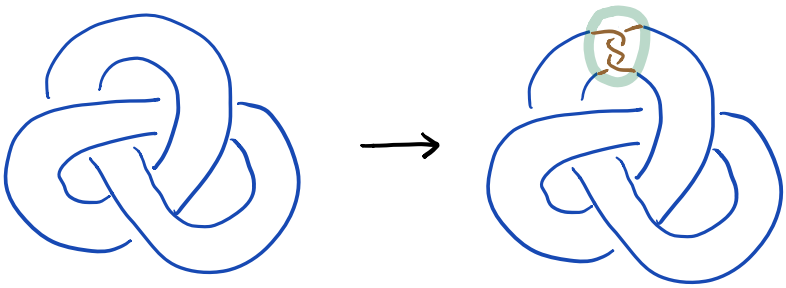


Figure 11 Inserting a tangle into the double of a trefoil.

5.9 Identifying satellites and certifying primeness

Our final step is to identify the exact structure of our 380 satellites. Our method here is simple: we predict which satellites we *expect* to see, explicitly construct them, and observe that these are the same as the 380 satellites in our census.

Our constructions are based on *tangles*. These are like knots, but instead of a closed loop of string, we have four immovable external endpoints connected by two pieces of string. See Figure 10 for some illustrations, or Adams [1] for a more thorough introduction.

The satellites we expect to see are formed by taking the double of the trefoil or figure eight knot (with 12 or 16 crossings respectively) and then inserting a tangle between two parallel strands, as shown in Figure 11.

The tangle we insert should use as few crossings as possible, should connect the two parallel copies of the trefoil or figure eight to form a single knot, and should not be the composition of a simpler tangle with another knot (as in Figure 10, right).

To build these tangles, we begin with Conway's *rational tangles*, a class of tangles that are fully classified and well understood [10]. We then hand-enumerate ten larger "templates", illustrated in Figure 12, and insert rational tangles into these templates so that the total number of crossings is ≤ 7 (thus the final satellites will have ≤ 19 crossings overall).

These tangles that we construct number 380 in total. From their constructions we know that they are prime satellites [12], and therefore must appear amongst the 380 satellites in our census. Computing their HOMFLY-PT polynomials shows that they are all distinct, and therefore (i) they are *precisely* the 380 satellites in our census, and (ii) the 380 satellites in our census are therefore prime. Since torus knots are also known to be prime, this certifies the primeness of all 394 non-hyperbolic knots.

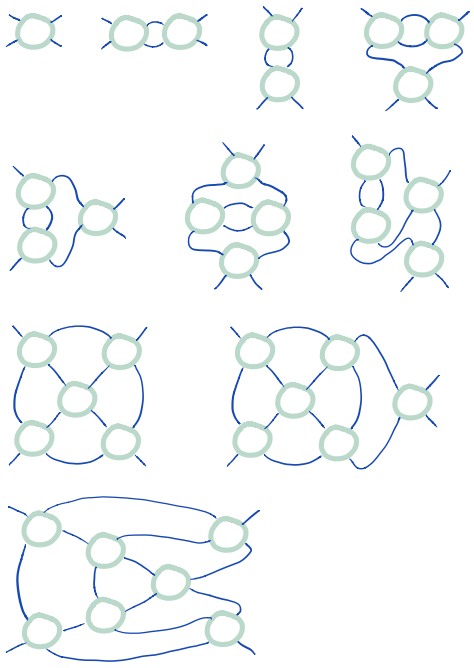


Figure 12 Templates into which we can insert rational tangles.

6 Stage two, continued: Finishing the hyperbolic case

Now we turn our attention to our 352 160 201 certified hyperbolic knots (these include the 18 extras that we picked up in section 5.7). We note that there is no need to certify primeness here, since primeness already comes as a consequence of hyperbolicity.

Extrapolating growth rates from smaller censuses suggests that our count should be close to the real number of distinct knots, and so our main task is to show that our knots are distinct. Of course we still expect to find duplicates, which we remove as they appear.

Throughout this section, we group our knots into classes, where knots in *different* classes have been certified as distinct, and knots in the *same* class might or might not be equivalent. Whenever a class shrinks down to exactly one knot, we can store that knot in the final census and forget about it in our computations here.

Throughout this section, we will count (i) the number of classes that still contain more than one knot, and (ii) the total number of knots that those classes hold.

Our intial state: 1 class and 352 160 201 knots.

6.1 Separating using HOMFLY-PT polynomials

Our first step is to compute the HOMFLY-PT polynomials of each knot, as described in section 5.6. We then sort the knots by their polynomials and use the results to split the knots into classes, one for each distinct polynomial.

As the number of crossings grows, the HOMFLY-PT polynomials become longer, with the result that simply writing all the polynomials consumes 230 GB of disk space. The subsequent sort operation was done using the GNU `sort` command, which only needed 40 GB of RAM but at the cost of doing significant work on the disk instead (which is much slower).

Our new state: 56 376 691 classes and 158 221 199 knots.

6.2 Subgroups of index 2–4

Next we turn to algebraic invariants: we can show that two knots are distinct by certifying that their complements have non-isomorphic fundamental groups.

Unfortunately, each fundamental group $\pi_1(\overline{K})$ is computed as a presentation involving generators and relations, and solving isomorphism problems on group presentations is notoriously hard (indeed, undecidable in general [26]). Instead we use the popular strategy of identifying invariants of these groups. For our application we enumerate all subgroups of index $k = 2, 3, 4$ and compute their abelian invariants, and we do the same for the *core* of $\pi_1(\overline{K})$ (the largest normal subgroup in $\pi_1(\overline{K})$). We use *GAP* [18] for our computations.

These are all computations that terminate and give exact output, though their running time increases exponentially (or worse) with the index k [30]. This is why we only use $k \leq 4$; we return to higher indices only once the number of knots remaining is substantially smaller.

Our new state: 42 091 807 classes and 115 086 342 knots.

6.3 Verified canonical triangulations

We now return to canonical triangulations, as computed in section 4.2. When used within *Sage* [29], *SnapPy* can in some cases *certify* its canonical triangulation. The underlying algorithm is based on certified interval arithmetic methods as originally used by *HIKMOT* [22] for similar topological applications.

Whether *SnapPy* and *Sage* are able to certify the canonical triangulation depends on many factors, such as numerical precision, the choice of underlying triangulation, and the geometric properties of the canonical cell decomposition. Nevertheless, if they *can* certify the canonical triangulations for two knots, we can just test the canonical triangulations for combinatorial isomorphism (a polynomial-time operation [5]): if they are isomorphic then the knots are duplicates, and if they are non-isomorphic then the knots are distinct.

When applied to our classes of potentially-equivalent knots, however, this process becomes more fragile. If *SnapPy* and *Sage* are able to certify canonical triangulations for *all* knots in a class, then we can completely resolve the class into duplicates and distinct knots. However, if the certification fails for just one knot K in the class, we cannot prove uniqueness of *any* of the knots in the class. We are still able to strip out duplicates where the canonical triangulations are isomorphic, but otherwise all knots in the class must stay – even if some of them are known to be pairwise distinct – because our problematic knot K could still be equivalent to any of them.

A technical problem with this process was a memory leak when using *SnapPy* with *Sage* (known to the *SnapPy* authors, but difficult to fix). As a result, the inputs had to be processed in small batches so the memory leaks did not accumulate too badly – an inconvenience for 115 million inputs, but nevertheless manageable.

Our new state: 7 086 classes and 21 221 knots, after removing 2 939 duplicates.

6.4 Exhaustive search for duplicates, 1 or 2 extra crossings

Next, within each class, we perform an exhaustive search for duplicates by trying all possible sequences of Reidemeister moves, allowing for at most two extra crossings at any stage. This is the same procedure seen in section 5.4 for non-hyperbolic knots.

Our new state: 3 727 classes and 12 715 knots, after removing 5 147 duplicates.[3]

[3] Note that each duplicate could potentially remove *two* knots from consideration, since removing the duplicate could leave a class with just one knot that is then moved into the final census.

6.5 Subgroups of index 5

We return again to algebraic invariants, but this time we process subgroups of index $k = 5$. Although the computations are enormously slower, we also have several orders of magnitude fewer knots to process than for indices $k \leq 4$, and so the computations remain managable.

Our new state: 479 classes and 1 040 knots.

6.6 Verified canonical triangulations, again

We return now to certified canonical triangulations with *SnapPy* and *Sage*, as seen before in section 6.3. Since our classes now contain fewer knots than they did before, it is reasonable to hope that more classes can now be resolved completely (i.e., they are not hampered by one problematic knot whose canonical triangulation cannot be certified).

Moreover, our total number of knots remaining is orders of magnitude smaller than before. We therefore make several attempts at certifying the canonical triangulation for each knot, by starting from many different retriangulations of the same knot complement.

The *SnapPy*/*Sage* memory leak remains a problem, and since we are making many attempts for each knot, our inputs need to be processed in much smaller batches. Again, since the total number of knots is much smaller than before, this is manageable.

We made several more runs through this process, beginning with 10 attempts per knot, and finishing with 300 attempts per knot.

Our new state: 45 classes and 105 knots, after removing 255 duplicates.

6.7 Subgroups of index 6

We return to algebraic invariants again, this time with index $k = 6$. As before, the computations are much slower again, but we have orders of magnitude fewer knots to process.

Our new state: 3 classes and 6 knots, all drawn in Figure 13 (each line shows one class).

6.8 Exhaustive search for duplicates, 3 extra crossings

As before, within each class we perform an exhaustive search for duplicates by trying all possible sequences of Reidemeister moves, this time allowing $\leq n + 3$ crossings at any stage.

This is enormously slow, but with only three classes remaining and a multithreaded implementation of the underlying breadth-first search, it remains (just) within feasibility.

Our new state: 1 class and 2 knots, after removing 2 duplicates. The remaining two knots are the green pair at the bottom of Figure 13.

6.9 Subgroups of index 7

For index $k = 6$, *GAP* was barely able to finish the computations and so for index $k = 7$ there is little hope. We therefore switch to *Magma* [2] for $k = 7$, since (in the author's experience) *Magma*'s running times are faster and more consistent. We did not use *Magma* until now because it is commercial software and we only had a license for a single machine, and so we could not parallelise the computations across a cluster.

Happily *Magma* was able to distinguish the final pair of knots, though even this depended on starting with the "right" group presentations. We tried three different presentations of the two groups (obtained by starting from different triangulations of the knot complements): for one pair the computations took an hour, for one pair they took closer to a day, and for one pair they did not finish after several days of running time.

Nevertheless, our new state: 0 classes and 0 knots. The census is complete!

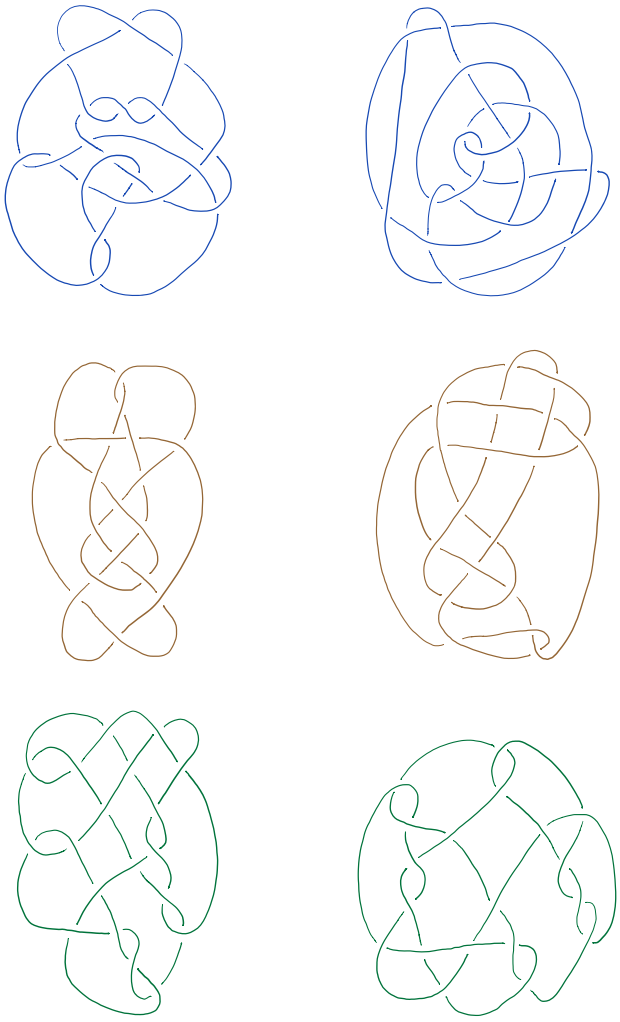


Figure 13 The three pairs of knots that remain after subgroups of index 6.

References

1. Colin C. Adams. *The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots*. W. H. Freeman & Co., New York, 1994.
2. Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993). doi:10.1006/jsco.1996.0125.
3. Gunnar Brinkmann and Brendan McKay. plantri, 2018. URL: http://users.cecs.anu.edu.au/~bdm/plantri/.
4. Benjamin A. Burton. Introducing Regina, the 3-manifold topology software. *Experiment. Math.*, 13(3):267–272, 2004.
5. Benjamin A. Burton. Simplification paths in the Pachner graphs of closed orientable 3-manifold triangulations. Preprint, arXiv:1110.6080, October 2011.
6. Benjamin A. Burton. The HOMFLY-PT polynomial is fixed-parameter tractable. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry*, volume 99 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages 18:1–18:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern. doi:10.4230/LIPIcs.SoCG.2018.18.

7 Benjamin A. Burton, Ryan Budney, William Pettersson, et al. Regina: Software for low-dimensional topology, 1999–2019. URL: http://regina-normal.github.io/.

8 Benjamin A. Burton and Melih Ozlen. A fast branching algorithm for unknot recognition with experimental polynomial-time behaviour. To appear in Math. Program., arXiv:1211.1079, November 2012.

9 Benjamin A. Burton and Stephan Tillmann. Computing closed essential surfaces in 3-manifolds. Preprint, arXiv:1812.11686, December 2018.

10 J. H. Conway. An enumeration of knots and links, and some of their algebraic properties. In *Computational Problems in Abstract Algebra (Proc. Conf., Oxford, 1967)*, pages 329–358. Pergamon, Oxford, 1970.

11 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 3rd edition, 2009.

12 Peter R. Cromwell. *Knots and links*. Cambridge University Press, Cambridge, 2004. doi:10.1017/CBO9780511809767.

13 Marc Culler, Nathan M. Dunfield, and Jeffrey R. Weeks. SnapPy, a computer program for studying the geometry and topology of 3-manifolds, 1991–2013. URL: http://snappy.computop.org/.

14 C. H. Dowker and Morwen B. Thistlethwaite. Classification of knot projections. *Topology Appl.*, 16(1):19–31, 1983. doi:10.1016/0166-8641(83)90004-4.

15 D. B. A. Epstein and R. C. Penner. Euclidean decompositions of noncompact hyperbolic manifolds. *J. Differential Geom.*, 27(1):67–80, 1988.

16 P. Freyd, D. Yetter, J. Hoste, W. B. R. Lickorish, K. Millett, and A. Ocneanu. A new polynomial invariant of knots and links. *Bull. Amer. Math. Soc. (N.S.)*, 12(2):239–246, 1985. doi:10.1090/S0273-0979-1985-15361-3.

17 David Futer and François Guéritaud. From angled triangulations to hyperbolic structures. In *Interactions Between Hyperbolic Geometry, Quantum Topology and Number Theory*, volume 541 of *Contemp. Math.*, pages 159–182. Amer. Math. Soc., Providence, RI, 2011.

18 The GAP Group. *GAP – Groups, Algorithms, and Programming*, 2019. URL: https://www.gap-system.org.

19 C. McA. Gordon and J. Luecke. Knots are determined by their complements. *J. Amer. Math. Soc.*, 2(2):371–415, 1989.

20 Robert C. Haraway, III. Determining hyperbolicity of compact orientable 3-manifolds. Preprint, arXiv:1410.7115, October 2014.

21 Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *J. Assoc. Comput. Mach.*, 46(2):185–211, 1999.

22 Neil Hoffman, Kazuhiro Ichihara, Masahide Kashiwagi, Hidetoshi Masai, Shin'ichi Oishi, and Akitoshi Takayasu. Verified computations for hyperbolic 3-manifolds. *Exp. Math.*, 25(1):66–78, 2016. doi:10.1080/10586458.2015.1029599.

23 Jim Hoste, Morwen Thistlethwaite, and Jeff Weeks. The first 1,701,936 knots. *Math. Intelligencer*, 20(4):33–48, 1998.

24 Louis H. Kauffman. State models for link polynomials. *Enseign. Math. (2)*, 36(1-2):1–37, 1990.

25 C. N. Little. Non-alternate ± knots. *Trans. Royal Soc. Edinburgh*, 39:771–778, 1900.

26 A. A. Markov. Insolubility of the problem of homeomorphy. In *Proc. Internat. Congress Math. 1958*, pages 300–306. Cambridge Univ. Press, New York, 1960.

27 Józef H. Przytycki and Paweł Traczyk. Invariants of links of Conway type. *Kobe J. Math.*, 4(2):115–139, 1988.

28 Igor Rivin. Euclidean structures on simplicial surfaces and hyperbolic volume. *Ann. of Math. (2)*, 139(3):553–580, 1994.

29 The Sage Developers. *SageMath, the Sage Mathematics Software System*, 2018. URL: https://www.sagemath.org/.

30 Charles C. Sims. *Computation with Finitely Presented Groups*, volume 48 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1994. doi:10.1017/CBO9780511574702.

31 William P. Thurston. Three-dimensional manifolds, Kleinian groups and hyperbolic geometry. *Bull. Amer. Math. Soc. (N.S.)*, 6(3):357–381, 1982.

32 Jeffrey R. Weeks. Convex hulls and isometries of cusped hyperbolic 3-manifolds. *Topology Appl.*, 52(2):127–149, 1993.