The 250 Knots with up to 10 Crossings

Andrey Boris Khesin

University of Toronto

The list of knots with up to 10 crossings is commonly referred to as the Rolfsen Table. This paper presents a way to generate the Rolfsen table in a simple, clear, and reproducible manner. The methods we use are similar to those used by J. Hoste, M. Thistlethwaite, and J. Weeks in [1]. The difference between our methods comes from the fact that [1] uses a more complicated algorithm to be able to find all the knots with up to 17 crossings, while our approach demonstrates a simpler way to find the knots up to 10 crossings. We do this by generating all planar knot diagrams with up to 10 crossings and applying several simplifications to group the knot diagrams into equivalence classes. From these classes, we generate the full list of candidate knots and reduce it with several sets of moves. Lastly, we use invariants to show that each of the 250 diagrams generated is distinct, proving that there are exactly 250 knots with 10 crossings or fewer. Though the algorithms used could be made more efficient, readability was chosen over speed for simplicity and reproducibility.

I Introduction

The Rolfsen table is the list of the 250 knots with 10 crossings or fewer. Here we attempt to generate it and prove its completeness by using a computer algorithm. This has been accomplished several times in the past. A notable example is [1], where J. Hoste, M. Thistlethwaite, and J. Weeks found all of the knots with up to 17 crossings. The methods we use are far less advanced, which allows us to effect a less intensive computation, finding the knots with up to 10 crossings, but use a simpler algorithm to do accomplish this.

Although it is possible to compute the Rolfsen table by hand, it is a rather tedious task. Our calculation is made possible by using a computer. To demonstrate a method of generating the Rolfsen table, we create a simple algorithm for finding all 250 knots with up to 10 crossings, partially sacrificing efficiency in the process.

We begin our reconstruction of the Rolfsen table by considering which knot diagrams could potentially be included in the table. There are only a finite number of ways to draw a knot diagram with a given number of crossings. Additionally, many of these knot diagrams are reducible, which means that they are equivalent to other knot diagrams with fewer crossings.

There are far more than 250 knot diagrams with up to 10 crossings, even after only irreducible knot diagrams are considered. The reason for this is that there are several moves that can transform one knot diagram into an equivalent one. Two knot diagrams are equivalent if and only if there exists a series of such moves that transforms one of the diagrams into the other.

Fig. 1. The 6 moves that we use to construct the Rolfsen table, as well as the second Reidemeister move. The letter R is used to denote a tangle with an appropriate number of strands. If the letter R appears in a different orientation it is because the move caused the corresponding part of the knot diagram to flip.
Andrey Boris Khesin

Right Left

Fig. 2. The right-handed trefoil and the left-handed trefoil. These knots are considered equivalent for our purposes as they are mirror images of each other. However, it is important to note that no series of moves can transform one of these into the other, so while they are not equivalent knots, we only include one of them in the Rolfsen table.

The first manner in which we simplify the list of knot diagrams is by eliminating knot diagrams that are mirror images of each other. For example, the right-handed and left-handed trefoils are not equivalent as it is impossible to turn one into the other (see Fig. 2). We only include one of the two in the Rolfsen table. The notation we use to represent a knot diagram does not encode the handedness of the knot so this is not an issue.

Fig. 3. An example of a composite knot diagram. This knot diagram can be cut along the dotted line into two knot factors, $T$ and $T'$. Both $T$ and $T'$ can be cut along one of their edges to create the pairs of ends $A$ and $B$, as well as $A'$ and $B'$, respectively. If $A$ is joined to $A'$ and $B$ to $B'$, the resulting knot is the knot composition of $T$ and $T'$, which are its knot factors. Knot diagrams that cannot be decomposed into two such knot factors are prime and are the kind of knot diagrams that we want to include in our tabulation.

For any two knot diagrams $T$ and $T'$, we cut $T$ at some point to create ends $A$ and $B$ and we cut $T'$ at some point to create ends $A'$ and $B'$. Joining $A$ to $A'$ and $B$ to $B'$ results in one larger knot, $R$. We say that $T$ and $T'$ are knot factors of $R$. The commutative operation of joining $T$ and $T'$ to create $R$ is called knot composition. Knots that cannot be decomposed into two knot factors other than themselves and the unknot are called prime. If they can be decomposed this way, they are called composite (see Fig. 3). We only include prime knots in our tabulation.

Lastly, we group knots into equivalence classes based on whether or not there exists a series of moves that transform one knot diagram into another (see Fig. 1). Out of each of these equivalence classes, we select one knot to include in our tabulation. Having done this, all that remains is proving that our list contains no remaining equivalent knot diagrams.

II MD Codes

The first thing we need to do is to establish is a way to efficiently represent a knot diagram with some sort of notation. This notation must be relatively easy for both humans and computers to work with.

In [1], a notation is used to represent an $n$-crossing knot diagram with $n$ integers. This notation is called Dowker notation. Its density and simplicity make it convenient for our purposes.

For an $n$-crossing knot diagram, its representation in Dowker notation is constructed as follows. We start by picking an arbitrary point on one of the knot diagram’s edges as well as an arbitrary direction along that edge. We then move along the knot diagram, moving along each edge, until we have traveled along all $2n$ edges and have returned to our starting point. Note that we will pass each crossing twice, once under and once over. Each time we pass a crossing, we consider the number of crossings that we have encountered so far and write that number down at the crossing that we are passing. In other words, when we encounter our first crossing, we write down the number 1 at that point. It is important to distinguish between writing the number on the upper or the lower strand of a crossing. If we passed the first crossing while traveling along the upper strand, we write down the number 1 on the upper strand and vice versa. Continuing, we would write down the number 2 at the next crossing we encounter. We would end up writing each number from 1 to $2n$ exactly once.

Note that since any two closed curves intersect in an even number of places, it follows
that the pair of numbers written at each crossing will contain one odd number and one even number. If this were not the case then we would be able to leave a crossing, travel in a closed loop, and come back to that crossing having encountered an odd number of crossings along the way, which is not possible.

The \( n \) pairs of numbers have no order, so sorting them in ascending order by comparing the odd value in each pair does not sacrifice any information. It then follows that the list of even values, sorted by their corresponding odd value, is sufficient to fully reconstruct the original list of pairs.

As an example, we show how we would find the Dowker notation for the trefoil. Note that the handedness of the trefoil is irrelevant. After labeling the trefoil, the pairs are (1, 4), (2, 5) and (3, 6) (see Fig. 4). We reorder the values in some of the pairs, in this case in (2, 5), to place the odd value first. Then the pairs can be ordered by their odd value to get (1, 4), (3, 6), and (5, 2). The original pairs can be reconstructed with the sequence (4, 6, 2) as there is a unique way of reestablishing the odd counterparts to each of the even numbers. Thus, (4, 6, 2) is the Dowker notation for the trefoil. Since this sequence contains only even numbers, storing half of each value works just as well and makes some computations easier. Therefore, we represent the trefoil by (2, 3, 1). We call the notation that stores half of each integer an MD code (M is for modified). The \( 2 \times n \) matrix of pairs is called an ED code (E is for extended). The ED code for the trefoil is \( \begin{pmatrix} 1 & 3 & 5 \\ 4 & 6 & 2 \end{pmatrix} \).

We will later refer to examining permutations. Since a MD code is a permutation of the numbers from 1 to \( n \), we can examine each such permutation to see if it encodes a viable knot diagram. Note the distinction between a permutation, one of the many ways of ordering the numbers from 1 to \( n \), and an MD code, a permutation of the number from 1 to \( n \) that encodes a particular knot diagram.

As described so far, this notation only tells us which strands cross which. What it does not tell us is the handedness of each crossing (see Fig. 5). In other words, the shape of the knot diagram can be reconstructed, but every crossing will effectively be blurred out, as it will not be clear which of the two strands in the crossing is the upper strand and which is the lower strand. To account for this, we declare that a crossing is positive if out of the two values that make up a crossing, the odd one corresponds to the upper strand of the crossing. If a crossing is not positive, it is negative and we indicate this by negating the even value in each negative crossing. For example, if a crossing is marked (17, 34) and the strand labeled 17 passes above the strand labeled 34, we leave the crossing as is. On the other hand, if the upper strand is marked 34, we denote the crossing by the pair (17, -34).

If we were to flip over a knot diagram and look at it from the back, all of the values in the MD code would change sign. To account for this, when necessary we negate all of the values in the MD code to make the leading term positive. As a result, every knot diagram with \( n \) crossings can be represented by a signed permutation of the numbers from 1 to \( n \).
III Alternating Knots

A subset of the knots we are trying to tabulate are called alternating knots. By determining which alternating knots should be included in our tabulation, we can simplify the task of determining the remainder of the list. For this reason, we start by determining which alternating knots should be included in our reconstruction of the Rolfsen table.

When we move along a knot diagram, labeling its edges to determine its MD code, we go over some strands and under others. If we always alternate between going over and under the strands we cross, we say that the knot diagram is alternating. We note that any minimal knot diagram that is equivalent to an alternating knot diagram will be alternating (see [2]). Thus, it makes sense to refer to alternating knots as this property is independent of our choice of knot diagram, as long as it is minimal. Any knot that is not an alternating knot is a non-alternating knot. We note that the MD code of an alternating knot will consist entirely of positive entries.

To generate all of the knot diagrams that might be included in our tabulation, we do not need to generate every possible knot diagram that there is. It suffices to first determine the list of alternating knots in our reconstruction of the Rolfsen table. Afterwards, we will construct the non-alternating knots from our finalized list of alternating knots.

We know that not all permutations of 10 values result in valid knots. Thus, some of these permutations must be eliminated from consideration. There are several criteria which we can use to determine which alternating knots should be included in our list. We will first define these criteria, then explain how to implement a test that verifies that they are satisfied. The alternating knot corresponding to a given permutation is included in our tabulation if and only if it meets all of the following criteria.

1. A knot diagram can produce different permutations depending on where one starts numbering and in which direction they proceed. There are $4n$ ways to choose both a starting point and a direction. A permutation is minimal if it is lexicographically smaller than or equal to all of the other $4n - 1$ possible permutations of the corresponding knot diagram. To satisfy this criterion, a permutation must be minimal.

2. The resulting knot diagram must be prime. Since a composite knot diagram can be split into two knot factors, we know that as we label the knot, we will have to go through all of the crossings of one of the knot factors before we move on to the other. This means that the values from 1 to $2n$ will be split into two consecutive subsequences since the values in each subsequence will be the labels of the crossings of one of the knot factors. In a permutation, this would be expressed as a set of $k$ pairs, all $2k$ of whose values form a consecutive subsequence. Thus, such a set must not exist in the ED code for the knot diagram to be prime (see Fig. 3). This also handily eliminates knot diagrams that contain a kink and could be simplified with the first Reidemeister move. The third Reidemeister move and the simplifying direction of the second Reidemeister move cannot occur in alternating knots (see Fig. 1).

3. The permutation must encode a diagram which is realizable. This means that there must be a way to draw the knot diagram in the plane without adding any intersections beyond the ones encoded in the permutation. The simplest permutation that fails this test is $(2, 4, 1, 5, 3)$. It is physically impossible to draw a knot diagram on a plane that would have a non-realizable permutation as its MD code.

4. The knot diagram must be minimal with respect to flyping. This means that the knot diagram’s minimal permutation must be lexicographically minimal over all of the permutations of knot diagrams that can be obtained from our original diagram by applying a sequence of flypes (see Fig. 1).

The first two conditions can be used to avoid testing all $n!$ possible alternating MD codes. If we arranged the $n!$ permutations lexicographically and went along checking each one, it will frequently be possible to skip checking up to $k!$ permutations at a time, where $0 \leq k < n$. 
Skipping these permutations is made possible as the first two criteria can be checked directly from the permutation. If a permutation that fails one of the first two criteria contains a string of values that make the permutation fail this criterion, then all permutations that contain the same string of values will also fail this criterion and do not need to be considered. Thus, as soon as we find such a permutation, we increment the last digit of the offending subsequence, thereby skipping $k!$ permutations ahead, where $k$ is the number of digits left in the permutation after the subsequence. In other words, given a leading subsequence that fails a criterion, we do not try to continue it with terms that we know will fail the same test.

For the first criterion, a permutation is not minimal if we can find a pair of values in our ED code which are numerically closer together than the leading term and its odd-valued pair, the number 1. So if there is an even value $x$ in the ED code which is closer to its paired odd value than the first number in the ED code is to 1, then all permutations with $x$ in the same position will not be minimal. This is because starting the enumeration of the knot diagram’s crossings at the one previously labeled $x$ would result in a smaller first element in the MD code, which is not allowed as the knot diagrams in the table must be minimal.

For the second criterion, a permutation is not minimal if we can find a subsequence of the knot that starts with this sequence will contain a knot factor, a trefoil in this case, and will not be prime.

In other words, if a knot is composite, its knot factors will contain consecutive strand labels. Since two knot factors can never cross, the crossings of a knot factor contain two elements of the subsequence that our permutation starts with. Thus, if the pairs of the values in a subsequence do not contain any values outside of that subsequence, the knot diagram is not prime.

The third condition is checked with the help of a modified graph planarity algorithm. If a 4-valent graph is constructed out of a knot diagram by replacing each crossing with a vertex and each edge of the knot with an edge in the graph, then typical planarity tests would frequently give false positives. There are 4 edges emanating from a crossing, but there are only 2 ways of arranging them in a valid manner in a knot diagram, but there are 6 ways of arranging 4 edges around a vertex. The reason for this is that a strand is not allowed to exit a crossing via an edge that is adjacent to its incoming edge. Strands must go directly across a crossing which means that the incoming and outgoing edges of a strand must be aligned opposite from each other.

We have not yet imposed any restrictions that would tell a graph planarity algorithm that such cases should not be considered. Permutations that fail this test do not form a planar knot diagram, but the graph that is created by making the same connections between vertices is planar. We can check that the graph formed by the permutation (2, 4, 1, 5, 3) is planar, yet such a knot diagram is impossible to draw, and is thus not realizable.

Similarly, if a knot diagram is composite, it is represented by several consecutive terms in the permutation, so all permutations obtained by rearranging the values that come after this sequence would also fail this test. Checking if a knot is prime was described above, we need to find a subsequence of the values from 1 to $2n$ such that all of the values’ pairs are just a reordering of the same subsequence. For example, if an MD code starts with (2, 3, 1, ...), it will not be prime. This is because the subsequence (1, 2, 3, 4, 5, 6) has pairs (4, 5, 6, 1, 2, 3). Since the second list is just a rearrangement of the first, any knot that starts with this sequence will contain a knot factor, a trefoil in this case, and will not be prime.

In other words, if a knot is composite, its knot factors will contain consecutive strand labels. Since two knot factors can never cross, the crossings of a knot factor contain two elements of the subsequence that our permutation starts with. Thus, if the pairs of the values in a subsequence do not contain any values outside of that subsequence, the knot diagram is not prime.

The third condition is checked with the help of a modified graph planarity algorithm. If a 4-valent graph is constructed out of a knot diagram by replacing each crossing with a vertex and each edge of the knot with an edge in the graph, then typical planarity tests would frequently give false positives. There are 4 edges emanating from a crossing, but there are only 2 ways of arranging them in a valid manner in a knot diagram, but there are 6 ways of arranging 4 edges around a vertex. The reason for this is that a strand is not allowed to exit a crossing via an edge that is adjacent to its incoming edge. Strands must go directly across a crossing which means that the incoming and outgoing edges of a strand must be aligned opposite from each other.

We have not yet imposed any restrictions that would tell a graph planarity algorithm that such cases should not be considered. Permutations that fail this test do not form a planar knot diagram, but the graph that is created by making the same connections between vertices is planar. We can check that the graph formed by the permutation (2, 4, 1, 5, 3) is planar, yet such a knot diagram is impossible to draw, and is thus not realizable.

![](image-url)

**Fig. 6**. The transformation applied to a knot diagram’s graph to determine if the knot diagram is planar. Each vertex in the 4-valent graph is replaced with 4 vertices connected to each other and to the original edges in a square. This makes the graph 3-valent and also serves as a proper indicator of the planarity of the knot diagram’s graph. The reason for this is that a graph should not be accepted as planar if there is a crossing where the two strands in the crossing enter and leave the crossing through adjacent edges. If this were to happen, the new graph would stop being planar as the square in the centre would become a non-planar bowtie.
To solve this problem, it is sufficient to replace each vertex with four vertices in a square to construct the modified graph of the knot diagram (see Fig. 6). This preserves the planarity of the two allowable configurations but bars the other four, as the square would be transformed into a non-planar bowtie shape. Thus, it suffices to use a regular graph planarity algorithm to check whether the modified graph of the knot diagram is planar. If it is not, the permutation fails the third criterion.

KnotGraph[k] gives the modified 3-valent graph
with 4 vertices in a square at each crossing of the
knot k, which is given in modified Dowker notation.

KnotGraph[k_MD] := KnotGraph[k] = Table[
  Array[{v, # - 1} -> v, Mod[#, 4] &, 4] |
  {v, 0} ->
  (Ordering[List[Abs[k]]][v], 1),
  {v, 2} -> (Ordering[List[Abs[k]]][
    Mod[v - 1, Length[k], 1]], 3),
  {v, Length[k]} // Graph;
KnotGraph[MD[2, 3, 4, 1]]

Finally, the fourth condition is checked by
using a graph searching algorithm to find
all knot diagrams that can be obtained from
a given knot diagram with a sequence of flypes
(see Fig. 1). From these, we keep only the
lexicographically minimal knot diagram. A
flype is represented in a permutation as a pair
and two disjoint subsequences of 1 to 2n. The
two subsequences are the strands that are the
part of the knot that gets flipped and the pair
is the crossing that gets moved to the other
side of the knot during the flype. Note that
these subsequences must satisfy several condi-
tions. The first is that all of the numbers
in these subsequences must be pairs of other
elements of the subsequences. This is much
like testing a knot for primality. The reason
for this condition is that the part of the knot
that is flyped must be like a two strand knot
factor, in the sense that it must not cross any
part of the knot outside of itself. Additionally,
these two subsequences, depending on
which way the strands run, must either start
or end with the values adjacent to those of the
earlier pair. Since that pair contains one odd
value and one even value, we can save time
by ignoring the cases where this would be
violated.

Flype[] gives a list of lists of all of the knots that can be
obtained by applying one flype to each knot of the list l.

Flype[] := {};
Flype[l_List :=
  Flype = Block[{a, c, e, n = Length[l][1]},
    p = List[Build[Abs[l][1]]] //
    (n'[Reverse[n]]'[2] & y = ()},
    Do[c = Mod[2 i - 1 + s[1]] Range[1, 2 n, 1];
      For[e = Max[Mod[Complement[p[0], c] - s[2][2 i - 1, j, 2 n, 1],
        e = Mod[s[2][2 i - 1 - 2 i[1, j]], 2 n, 1],
        e = c | Mod[2 i - 1, 2 n, 1][s[2][1] Range, 2 n, 1];
        If[Sort[p][c] = c,
          (*A flype can be made
          with the given settings.*)
          y = Join[y, (1, Convert /@ Mod[
            a = Abs[n] +
            Which[Mod[s[1]] [a - 2 i + 1],
              2 n, 1] <= c, -s[1]],
            Mod[s[2][2 i - 1, j]],
              2 n, 1] <= c, -s[2],
            a = 2 i - 1, s[1][0,
              a = 2 Abs[1][1, j], s[2][e,
              True, 0],
              2 n, 1] Sign & n &
              // Map[n, Build /@ {1, 3}]]])]
            {i, n},
            {s, ((1, 1), (1, -1), (-1, 1))},
            {0, 2,
              Mod[s[1][2 i - 1, j, 2 n, 1] - 1, 2 n, 1][j - 1];
              KnotSort /@ If[Dimensions[y = (2),
                y, (y]]})]];
  Flype[MD[2, 5, -7, -8, 1, -9, -10, -3, -4, -6]]

{(MD[2, -6, -7, 9, 1, -8, -10, -4, -3, -5]),
  MD[2, 5, -7, -8, 1, -9, -10, -3, -4, -6]),
  MD[2, 3, 5, 10, -8, 1, 2, -9, -4, 7],
  MD[2, 5, -7, -8, 1, -9, -10, -3, -4, -6]),
  MD[2, 5, -7, -8, 1, -9, -10, -3, -4, -6],
  MD[2, 5, -7, -8, 1, -9, -10, -3, -4, -6])}
Without the flyping condition, we have generated a list of all candidate knot diagrams. This is a list of all diagrams that encode alternating knots and all appear distinct at first glance. We will use this list later when we will want to examine all valid knot diagrams as opposed to those that are necessarily distinct.

CandidateKnots@n gives the sorted list of all irreducible planar minimal alternating knot diagrams with n crossings.

CandidateKnots@0 := {MD[]};
CandidateKnots@n_Integer :=
  AlternatingKnots@n =
  First/@KnotSort/@
  Join@Table[Sort[k <-> #] & /@
    KnotSort[Flype@{k}[]]],
    {k, CandidateKnots@n}]
  // Graph
  // ConnectedComponents
  // KnotSort;
PassReducible@k gives True if the knot \( k \), which is
given in modified Dowker notation, is reducible with a
2,1-pass move or a 3,2-pass move, and False otherwise.

ValidKnots\[7\] := 
ValidKnots@0 := {MD[]};
ValidKnots@n_Integer := 
\{Select[Join @@ Table[
MD@@ (c List@@ #)&/@CandidateKnots@n,
\{c, Tuples\[{1,-1}, n\] \[;; 2 n-1 \}\]
\& Minimal@#&\]
\\KnotSort\[\\];
\}

KnotSort@l sorts the knots of the list \( l \) where
alternating knots always come first and all
further order is determined lexicographically.

V Finding Equivalent Diagrams

Canonically, alternating knots precede non-alternating knots in the Rolfsen table.
We maintain the same pattern, ordering the
knots within each of the two sets lexicographically. Additionally, knots with a lower
crossing number always precede those with a higher crossing number.

KnotSort@l_List := 
KnotSort@l = Sort[1, If[Length@#1 = Length@#2,
If[#1 === Abs/@ #1,
#2 =!= Abs/@ #2 \[Order@#1, #2\] \[\] ⩵ 1,
#2 =!= Abs/@ #2 \[Order@#1, #2\] \[\] ⩵ 1,
Length@#1 < Length@#2\]&

KnotSort@MD[2, 4, 5, 6, 1, 7, 3], MD[2, 4, 6, 1, 7, 3, 5],
MD[2, 4, 6, 5, 1, 7, 3], MD[2, 4, 6, 7, 1, 3, 5],
MD[2, 5, 6, 7, 1, 4, 3], MD[2, 5, 7, 6, 1, 3, 4],
MD[2, 5, 7, 6, 1, 4, 3], MD[3, 5, 6, 7, 1, 2, 4],
MD[3, 5, 6, 7, 2, 1, 4], MD[4, 5, 6, 7, 1, 2, 3] \})
TwoPass@k gives all the knots that can be obtained by applying one 2-pass to the knot \( k \), which is given in modified Dowker notation.

TwoPass@k MD :=
TwoPass@k = Block[{{a, c, n = Length@k, p = List @@ Build@k //
{\{n\} || (Abs@Reverse[# Sign@#])'[2] &,
v, y = {}}},
Do[v = Abs@p[Mod[{i, i + 1}, 2 n, 1]]; If[Sort@
Sign@p[Mod[{i, i + 1}, 2 n, 1]] == {-1, 1},
Do[If[Total@Mod[1, 2] == 2,
c = Range@0
Partition[1 + {1, -1, 1, -1, 2}]; If[-MemberQ[Join @@ c, i],
l = RotateLeft@1;
c = Mod[Range@0 Partition[
1 + {1, -1, 1, 2 n - 1, 2}, 2 n, 1]]; If[Length[Join @@ c] < 2 n - 4
\v] Join @@ c = Abs@p[Join @@ c] ||
Mod[{i, i + 1}, 2 n, 1],
(* A 2-pass can be made with the given settings. *)
AppendTo[y, Convert@
Build@k /. x_Integer ->
PassMapping[v, l, p, c, n, Abs@x, 1]]]]],
{1, Sort@Join[\(\{x, y\}\) & /@
Subsets[Delete[Range[2 n],
Mod[{{i, i + 1}}', 2 n, 1]], {2}])))),
{1, 2 n}];
KnotSort[Minimal@y]]];
TwoPass@MD[3, -5, -9, 7, -1, -8, 10, 4, -2, 6]
{MD[3, 8, 6, 7, -9, 2, 10, 1, -4, -5]}

To find equivalent knot diagrams, we implement a graph searching algorithm. We first need to build our graph recursively by adding on subsequent layers of knot diagrams. We start with a graph \( \Gamma_0 \) consisting of the set of vertices \( V_0 \) and edges \( E_0 \). We define \( V_0 \) as the set of those 1176 knot diagrams that are not immediately reducible. For every natural number \( i \), we define \( V_i \) as the union of \( V_{i-1} \) and the set of knot diagrams that can be obtained by applying one crossing number-preserving move to a knot diagram in \( V_{i-1} \). We do not need to define our edges recursively. For any non-negative integer \( i \), we define \( E_i \subset V_i \times V_i \). For any two knot diagrams \( v_1 \) and \( v_2 \) in \( V_i \) we include the undirected edge \( (v_1, v_2) \) in \( E_i \) if we can apply a crossing number-preserving move to \( v_1 \) and obtain \( v_2 \) (see Fig. 1). Lastly, \( \Gamma_i \) is simply the set of vertices \( V_i \) and set of edges \( E_i \).

Since there are finitely many knot diagrams with a given number of crossings, there must exist an integer \( i \) such that \( \Gamma_i \) is equivalent to \( \Gamma_{i+1} \), at which point the graph will cease to change. We then take \( \Gamma_i \) to be our graph. Each connected component of the graph consists of a set of equivalent diagrams, all representing the same knot.

At this point we return to the earlier concern that this graph might contain some reducible knot diagrams. Any reducible knot diagram that has not yet been removed is not immediately reducible, which means that it cannot be reduced with a pass move or the second Reidemeister move. However, all such diagrams are equivalent to other diagrams which are immediately reducible. Thus, we check to see if any of the knot diagrams in a connected component are immediately reducible. If at least one is, we need to remove the entire component from the graph. To do this, we create a graph \( \Gamma' \) which is a subgraph of \( \Gamma_i \) and contains only the edges and vertices of the components which do not contain any immediately reducible knot diagrams. After this, \( \Gamma' \) does not contain any reducible knot diagrams.

We now must generate our list of knots from the graph \( \Gamma' \). To do this, we take the lexicographically smallest knot diagram from each connected component of \( \Gamma' \). As previously mentioned, we get 54 such knots. Combined with the 197 alternating knots, this gives us 251 total knots. However, just because we have applied a variety of moves to construct edges in our graph does not mean that we are done. It is possible that there are equivalent knot diagrams in the graph \( \Gamma' \) between which we were unable to find a sequence of moves out of our set. Thus, all we know is that there are no more than 251 knots with 10 crossings or fewer. Our lower bound is currently 200, as we know that our alternating knots are distinct and that we have at least one non-alternating knot with each of 8, 9, and 10 crossings.

The reason that we may have more knot diagrams than we should is because we are restricting ourselves to using flypes, 2-passes, and the third Reidemeister move to find equivalent knot diagrams. Reidemeister’s original theorem has the consequence
that if two minimal knot diagrams are equivalent and it is impossible to transform one into the other by repeatedly applying the third Reidemeister move, the only option left available to us is to first add crossings using one of the first two Reidemeister moves, and proceed from there. We are in a similar position because to show that two of our knot diagrams are equivalent, we would have to turn them into more complicated equivalent knot diagrams. However, though there are relatively few ways to apply the third Reidemeister move to a knot diagram, there are many ways of adding a kink to one.

Our next step is to figure out which pairs of knot diagrams in our list might be equivalent. Since there are so many ways of adding kinks, checking all of our knot diagrams for equivalence this way would take a long time. Thus, we would like some way to establish with certainty that some of our knot diagrams cannot be equivalent to any others. We do this by using invariants.

VI INVARIANTS

All invariants are functions from the space of knot diagrams to an arbitrary target space. The property that an invariant must satisfy is that the images of two equivalent knot diagrams must be equal under an invariant. This allows us to show that two knot diagrams on which an invariant produces a different value are knot diagrams of two different knots.

The invariant condition is easy to satisfy as we could choose the image of the invariant to have unit magnitude. For example, we could state that for any knot diagram, our carefully crafted invariant produces a value of 0. However, such an invariant is useless as we want to be able to show that some knot diagrams are distinct. Thus, we need invariants that produce the same value for non-equivalent knot diagrams as rarely as possible. The degree to which an invariant accomplishes this is typically called its strength, where weak invariants often produce the same values for different knot diagrams and vice versa. Often, stronger invariants require more time to compute, which is why it is useful to have several invariants. We use them in increasing order of strength so that the most difficult computations only have to be done for a few knot diagrams, those between which the weaker invariants were unable to distinguish.

The weakest invariant that we have available to us is crossing number. We know that all of our knots are non-reducible and we also know that all of our alternating knots are distinct. This immediately reduces our task to simply calculating invariants on non-alternating knots that all have the same crossing number.

After this initial step, we use two different invariants to complete the task. Our initial simplifications would typically be considered too crude to be deemed invariants except in the most technical of circumstances. Our first invariant, the Jones polynomial, is fast and fairly strong and the second, the number of colourings, is slow but even stronger.

Invariants[a, t] gives the new values of a and t after one of each set of equivalent knots in a is moved to t.

Invariants[a_List, t_List] := Invariants[a, t] = Block[{l = a, r = t},
Do[If[(i = i/@l) ≠ {},
   r = Select[{l, ...}],
   l = Complement[l, r]],
   {i, {If[# === Abs/@ #,#, 0]&,
   JonesPolynomial, Colourings[#,#]&}}];
{l, r};
Invariants[{MD[2, 4, 5, -7, 1, -8, -9, -3, -10, -6],
   MD[2, 6, -8, 7, -9, 1, 4, -10, -5, -3],
   MD[3, -5, -7, 8, -1, 9, -2, 10, 4, 6],
   MD[3, 5, 7, 8, 9, 2, -10, 1, 4, -6]};
{l, r};
Invariants[{MD[2, 6, -8, 7, -9, 1, 4, -10, -5, -3],
   MD[3, -5, -7, 8, -1, 9, -2, 10, 4, 6]},
   MD[2, 4, 5, -7, 1, -8, -9, -3, -10, -6],
   MD[3, 5, 7, 8, 9, 2, -10, 1, 4, -6}]}

VII PLANAR DIAGRAM NOTATION

To find the Jones polynomial of a given knot diagram, the knot diagram must be written in planar diagram notation as opposed to an MD code. To find this notation, it is necessary to determine the handedness of each of the knot diagram’s crossings (see Fig. 5). All we know about a knot diagram’s crossings from an MD code is which strand passes above or below. We do not know the handedness of each crossing. As there are 2 types of crossings, there are 2^n possible ways to
set the handedness of a knot diagram’s crossings. Since the only knot diagrams being considered are realizable, it is known that at least one of these $2^n$ crossing orientations will make the knot diagram planar.

Since we are trying to compute polynomials of knot diagrams where $n \leq 10$, we have that $2^n \leq 1024$, which is, computationally speaking, a small number. For this reason, we can exhaustively iterate through the $2^n$ crossing orientations until we find one that creates a planar knot diagram.

Fig. 7. The transformation applied to the knot diagram’s graph to determine whether or not the given configuration of crossings makes the knot diagram planar. We take each vertex in the 4-valent graph and replace it with 4 vertices connected to each other and to the original edges in a diamond. However, the connections to the adjacent vertices have been expanded to be a pair of parallel strands. This serves as a proper indicator of the planarity of the knot diagram’s graph with the given crossing configuration. The reason for this is that a graph should not be accepted as planar if there is a crossing where the two strands in the crossing enter and leave the crossing through adjacent edges. The new graph would stop being planar if this were to happen since the diamond in the centre would become a non-planar bowtie. Additionally, the graph should not be accepted as planar if the handedness of the crossing is changed. If this happens, the ribbons would twist and stop being planar.

To test if the knot diagram with given crossing orientations is planar, we apply the knot diagram planarity replacement from before, but replace each outer edge with a ribbon, a pair of parallel edges, to only allow 1 of the 6 edge configurations (see Fig. 7).

Earlier, the square replacement remained planar for either of the two ways of arranging the edges of the crossing so that the strands enter and exit the crossing through opposite edges (see Fig. 6). Now, we wish to exclude one of these two configurations. We do this by arranging the strands into the configuration we desire and changing the incoming edges into pairs of edges. Now, whenever the strands do not exit and enter through opposite edges, the graph will not be planar, just as before and for the same reasons. More importantly, when the handedness of the crossing changes, a twist will be added to two of the ribbons, making the graph non-planar. Thus, all we need to do is search through all possible sets of crossing orientations until we find one for which this modified graph is planar.

\[
\begin{pmatrix}
k & j \\
l & i
\end{pmatrix}
\]

Fig. 8. A right-handed crossing labeled in planar diagram notation. The lower incoming edge is labeled $i$ and then the remaining three are labeled $j$, $k$, and $l$, proceeding counterclockwise from $i$. The crossing is labeled as $X_{i,j,k,l}$.

Every crossing is represented in planar diagram notation as $X_{i,j,k,l}$ (see Fig. 8). Here, $i$ is the index given to the lower incoming edge and then $j$, $k$, and $l$ proceed counterclockwise. The knot diagram is then written as the product of its crossings in planar diagram notation. For example, the left-handed trefoil (see Fig. 4) is written as $X_{1,3,4,5}X_{3,6,4,1}X_{5,2,6,3}$.

ToPD@k gives a planar diagram notation for the knot $k$, which is given in modified Dowker notation.

\begin{verbatim}
ToPD@MD[3,-5,-9, 7,-1,-8, 10, 4,-2, 6]
PD[X 1,6,2,7, X 10,4,11,3, X 15,8,12, X 2,10,3,9, X 16,11,12, X 13,11,14,20, X 15,9,16,6, X 4,18,5,17, X 19,13,20,12]
\end{verbatim}

Once we can transform knot diagrams into planar diagram notation, we can compute their Jones polynomials.
The Jones polynomial of a knot diagram is computed from the product of the knot diagram’s crossings.

Smoothings for a Right-Handed Crossing

![0-smoothing and 1-smoothing](image)

Fig. 9. The 0 and 1-smoothings of a right-handed crossing. The smoothings are comprised of two strands with no directionality. If every crossing in a knot diagram is replaced by a smoothing, the result is an unlink as the knot diagram will be devoid of any crossings or ends. The 0-crossing is formed by connecting each of the two ends of the lower strand of the crossing to the ends of the upper strand that are next to them in the counterclockwise direction. For the 1-smoothing, the direction is clockwise. The 0-smoothing for a right-handed crossing is identical to the 1-smoothing for a left-handed crossing and vice versa.

Every crossing can be smoothed in two distinct ways (see Fig. 9). By smoothing a crossing in a particular manner, the polynomial of that smoothing is multiplied by a coefficient of either $A$ or $B$ for the 0 and 1-smoothings, respectively. Since each smoothing is actually a coefficient multiplied by the two non-intersecting strands of the smoothing, a strand stitching operation is applied to turn a product of $n$ smoothings into an unlink of several components.

A strand stitching operation satisfies the property that the product of two strands that share an endpoint, such as the strand from $p$ to $q$, $(p, q)$, and the strand $(q, r)$, will be equal to one strand running between their non-common endpoints, $(p, r)$ in this case.

The final result will always be the product of several strands that are closed loops of the form $(p, p)$. Each of these components of the link is given a coefficient of $d$ and thus the result is a polynomial in $A$, $B$, and $d$. What we have defined so far is called the Kauffman bracket of a knot diagram $X$, and it is denoted $\langle X \rangle$. We note that $\langle \bigcirc \rangle = d$ and $\langle \varnothing \rangle = 1$, where $\bigcirc$ and $\varnothing$ represent the unknot and the empty knot, respectively. Using this notation, a formula for the smoothings of a crossing can be written.

\[
\langle \bigcirc \rangle = A \langle \bigcirc \bigcirc \rangle + B \langle \bigcirc \bigcirc \rangle \quad (1)
\]

\[
\langle \bigcirc \rangle = A \langle \bigcirc \bigcirc \rangle + B \langle \bigcirc \bigcirc \rangle \quad (2)
\]

A given smoothing is a 0-smoothing if the incoming end of the lower strand is connected to the next end going counterclockwise around the crossing, in other words, the nearest end on its right. If it is connected to the end on its left, the resulting smoothing is a 1-smoothing.

Thus, $\langle \bigotimes \rangle$, the Kauffman bracket of the right-handed trefoil can be evaluated. Note that this trefoil is right-handed so we will only need (1). There are two ways to smooth each crossing so there are eight ways to smoooth the three crossings altogether. Two of these ways are applying three 0-smoothings and three 1-smoothings. The other six cases are not all distinct, since there are three identical ways of applying either one or two 0-smoothings. Thus, each of the cases in these sets have the same bracket value, which is how the bracket of the trefoil is expanded.

\[
\langle \bigotimes \rangle = A^3 \langle \bigotimes \rangle + 3A^2B \langle \bigotimes \rangle + 3AB^2 \langle \bigotimes \rangle + B^3 \langle \bigotimes \rangle \quad (3)
\]

Unsurprisingly, (3) looks a lot like an application of the binomial theorem. However, that is only because the trefoil is rotationally symmetric. In the general case, the bracket will not have as many like terms.

By counting the number of components in each unlink, the remaining brackets are evaluated with the corresponding power of $d$.

\[
\langle \bigotimes \rangle = A^3d^2 + 3A^2Bd + 3AB^2d^2 + B^3d^3 \quad (4)
\]

To make the Kauffman bracket invariant over the second Reidemeister move, we need to set $d + A^2 + B^2 = 0$ and $AB = 1$. We find that these relations make the Kauffman bracket invariant over the third move as well. This means that to show that this is actually...
an invariant, it suffices to show that the Kaufmann bracket is invariant over the first Reidemeister move. We find that in its current form, this is not the case.

\[ \text{SetAttributes[Strand, Orderless];} \]
\[ \text{Strand[i_, j_]} := \text{Strand[i, k_]} := \text{Strand[i, k_]} := \text{Strand[i, k_]} := \text{-q}^{1/2} - \text{q}^{-1/2}; \]
\[ \text{Strand[i_, j_]} := -\text{q}^{1/2} - \text{q}^{-1/2}; \]

To make the Kaufmann bracket invariant over addition and removal of kinks, the whole polynomial must be multiplied by a coefficient of \((-A)^{-3w}\) where \(w\) is the writhe of the knot diagram, which is the difference between the number of right-handed and left-handed crossings in the knot diagram.

\[ \text{Writhe@k} \text{ gives the writhe, the difference between the number of right-handed and left-handed crossings of the knot k, which is given in modified Dowker notation.} \]

\[ \text{Writhe@MD[3, -5, -9, 7, -1, -8, 10, 4, -2, 6]} \]
\[ \text{11} \]

The resulting polynomial will have a factor of \(d\) in every component as every unlink that we can get by smoothing the knot diagram has at least one component. Thus, the polynomial is normalized by dividing it by \(d\). Lastly, what we have obtained will always be a polynomial in \(A^4\) so the rule \(A = q^{1/4}\) is applied to make the result a Laurent polynomial in \(q\) (see [2]).

For the trefoil, these substitutions allow us to transform our equation into a simpler form. We get that the writhe, \(w\), is equal to 3. This means that the Jones polynomial for the right-handed trefoil needs to be multiplied by \(-A^3\). Applying \(d = -A^2 - B^2\) and \(B = A^{-1}\), we get the Jones polynomial of the trefoil.

\[ J(\mathcal{O}) = -A^{16} + A^{12} + A^{-4} \] (5)

Since the Jones polynomial of the mirror image of a knot diagram is the Jones polynomial of the original knot diagram with \(q\) replaced by \(q^{-1}\), the minimal of these two polynomials is taken as the value of the invariant for that knot diagram.

Applying the \(q\) substitution will yield the final version of the Jones polynomial for the right-handed trefoil. However, the left-handed trefoil has a smaller Jones polynomial by degree so we state that the Jones polynomial for the trefoil is the Jones polynomial for the left-handed trefoil.

\[ J(\mathcal{O}) = -q^4 + q^3 + q^{-1} \] (6)

JonesPolynomial@k gives the Jones polynomial of the knot \(k\), which is given in modified Dowker notation.

\[ \text{JonesPolynomial@MD} := \text{JonesPolynomial@MD} = \left( (-q^{1/4}) \text{Writhe@k Expand[Times@@ToPD@k /. \{X a_,b_,c_,d_ \to \text{Strand}[a, b] Strand[c, d] q^{-1/4} + Strand[a, d] Strand[b, c] q^{1/4}\}} \right) \]
\[ \text{//Sort}[1]]; \]

We find that the Jones polynomial shows that every knot diagram out of our 251 with 9 crossings or fewer is distinct. This is because all of the non-alternating knots with fewer than 10 crossings have distinct values for their Jones polynomial. Among the knots with 10 crossings, we find two pairs of knot diagrams with the same Jones polynomial. As we have dramatically reduced the list of diagrams we are unsure about, we can now apply our more powerful invariant at little cost.

Note that this raises our lower bound to 249. This is because there can be at most two extra knots in our list as the Jones polynomial only found two pairs of knots whose Jones polynomial was not distinct.

**IX Knot Colourings**

We have two pairs of knots in our list that could be equivalent. We need to determine if the knot diagrams in either pair are distinct. As we have very few knot diagrams to analyze, we can spend some additional time computing a more complicated invariant, in exchange for it being able to distinguish between our knot diagrams. This invariant is
the number of *colourings* of the knot diagram with elements of the permutation group $S_m$, for some $m$. Such a colouring is an assignment of permutations of $m$ elements to edges of the knot diagram such that these permutations satisfy a particular set of conditions. The number of such colourings is invariant over the three Reidemeister moves, making it invariant over all equivalent knot diagrams.

If two knot diagrams are equivalent, the number of colourings using elements of $S_m$ will be the same for all natural numbers $m$. To show that two knot diagrams are not equivalent, it is sufficient to find a value of $m$ such that the the invariant produces a different value for the two knot diagrams. Thus, the number of ways to colour both knot diagrams using elements of $S_m$ must be different.

It would be incorrect to simply count the number of ways that various values of $S_m$ can be assigned to all $2n$ edges of the knot diagram. To be an actual invariant, the assignments must satisfy two conditions. The reason for that is that the values assigned to edges along an arc must be the same. Thus, if we have a crossing $X_{i,j,k,l}$, we label the values in $S_m$ that we assign to each edge as $\sigma_i$, $\sigma_j$, $\sigma_k$, and $\sigma_l$ starting from the incoming lower edge and proceeding counterclockwise. From this, the two relations that these four values have to satisfy are constructed.

We know that the permutations must be equal along any arc. Thus, the two edges of the top strand have the same permutations.

$$\sigma_j = \sigma_l \quad (7)$$

The second criterion that our permutations must satisfy is as follows. The signed product of the permutations assigned to the four edges around a crossing must be the identity permutation. To clarify, we start by picking an arbitrary sign convention, which in our case is that inward pointing edges are given a positive sign. Then, we move around a crossing in an arbitrary direction from an arbitrary starting point, which in our case are counterclockwise from the edge labeled $i$. Inverting the permutations for the negative, and thus outward pointing, edges, we construct an equation that positive crossings must satisfy. Note that in a positive crossing, edges $j$ and $k$ point outwards.

$$\sigma_i \sigma_j \sigma_k^{-1} \sigma_l^{-1} = e \quad (8)$$

Where:

$e$ = the identity permutation with $m$ elements

For a negative crossing, we switch the signs of $j$ and $l$.

$$\sigma_i \sigma_j \sigma_k^{-1} \sigma_l^{-1} = e \quad (9)$$

By putting together (7) and (8), we get an equation for each positive crossing.

$$\sigma_k = \sigma_j \sigma_i \sigma_j^{-1} \quad (10)$$

So we can find $\sigma_k$ by finding the permutation conjugation of $\sigma_i$ and $\sigma_j$. We also get an equation for negative crossings.

$$\sigma_k = \sigma_j^{-1} \sigma_i \sigma_j \quad (11)$$

Thus, for negative crossings, $\sigma_k$ is the permutation conjugation of $\sigma_i$ and $\sigma_j^{-1}$.

PermutationConjugation[i, j] gives the conjugation of the permutations $i$ and $j$ by evaluating $jj^{-1}$

PermutationConjugation[i_List, j_List] := PermutationConjugation[i, j] = PermutationProduct[j, i, InversePermutation@j];

PermutationConjugation[{4, 1, 5, 2, 3}, {3, 5, 4, 1, 2}] {2, 1, 5, 3, 4}

We note that since we are taking the permutation conjugation of $\sigma_i$ with $\sigma_j$, then $\sigma_i$ and $\sigma_k$ will have the same cycle lengths. As any two edges across a crossing have the same cycle lengths, and since we can follow the path of the knot by going through each crossing, one by one, never changing cycle lengths, then all of the values for the edges must have the same cycle lengths. This gives us a lot more information.

Whereas before we would have had to map edges to $S_m$ and count the total number of colourings, now they can be mapped to a subset of $S_m$. If all the elements of this subset have the same cycle lengths, then the number of colourings for each such subset can be counted independently. Thus, instead of end-
We create a graph with vertices $V = \mathcal{P}(S)$, where $\mathcal{P}$ is the power set function. Note that $|V| = 2^n$, which for $n = 10$ is not too large. For each crossing, (10) tells us that knowing $\sigma_j$ as well as either $\sigma_i$ or $\sigma_k$, is sufficient to determine the other one, either $\sigma_k$ or $\sigma_i$, respectively. We represent this in the graph with a directed edge. From every vertex with set $T \subset S$ where $T$ contains both $i$ and $j$, we draw a directed edge to the vertex with set $T \cup \{k\}$. Similarly, we draw an edge from each vertex whose set $T \subset S$ contains both $j$ and $k$ to the vertex with set $T \cup \{i\}$. An edge from vertex $v_1$ to vertex $v_2$ represents the fact that by knowing the permutations for all of the edges with indices in the set $v_1$, we can determine the permutations for all of the edges with indices in the set $v_2$ by applying (10) or (11) once.

We consider the vertices from which there is a path to $S$, the vertex containing all of the indices. If there is a path from a vertex $v$ to the vertex $S$, then we know that being given the permutations for the edges whose indices are in $v$ is sufficient to determine all of the permutations of the knot diagram. Thus, to find our set of generators, we choose the vertex $v$ from which there is a path from $v$ to $S$ such that $|v|$ is as small as possible. To find the sequence of edges whose permutations can be determined, we examine any path from $v$ to $S$ and choose that as our order.

EdgeSequence@k gives a list of the strands in the order that they should be calculated from the generators, which are returned as a list as the first element of the result, of the knot $k$, which is given in modified Dowker notation.

We can determine the rest of the 20 permutations. It is immediately clear that $n$ edges can be derived from the other $n$ by using (7). Thus, we are only interested in finding which of the other $n$ can generate all of the permutations. What we are effectively doing is writing out the indices from 1 to $2n$ and striking out all those indices which can be determined from those that remain. For each crossing, we examine the edges that make up the upper strand and remove the one with the larger index from our list. After we have done this, we have $n$ remaining indices. We will call the set of these indices $S$. For some perspective, when $n = 10$, there are between 3 and 5 generators which can determine the rest of the 20 permutations. We consider the vertices from which there is a path to $S$, the vertex containing all of the indices. If there is a path from a vertex $v$ to the vertex $S$, then we know that being given the permutations for the edges whose indices are in $v$ is sufficient to determine all of the permutations of the knot diagram. Thus, to find our set of generators, we choose the vertex $v$ from which there is a path from $v$ to $S$ such that $|v|$ is as small as possible. To find the sequence of edges whose permutations can be determined, we examine any path from $v$ to $S$ and choose that as our order.

EdgeSequence@k gives a list of the strands in the order that they should be calculated from the generators, which are returned as a list as the first element of the result, of the knot $k$, which is given in modified Dowker notation.

We can determine the rest of the 20 permutations. It is immediately clear that $n$ edges can be derived from the other $n$ by using (7). Thus, we are only interested in finding which of the other $n$ can generate all of the permutations. What we are effectively doing is writing out the indices from 1 to $2n$ and striking out all those indices which can be determined from those that remain. For each crossing, we examine the edges that make up the upper strand and remove the one with the larger index from our list. After we have done this, we have $n$ remaining indices. We will call the set of these indices $S$. For some perspective, when $n = 10$, there are between 3 and 5 generators which can determine the rest of the 20 permutations. We consider the vertices from which there is a path to $S$, the vertex containing all of the indices. If there is a path from a vertex $v$ to the vertex $S$, then we know that being given the permutations for the edges whose indices are in $v$ is sufficient to determine all of the permutations of the knot diagram. Thus, to find our set of generators, we choose the vertex $v$ from which there is a path from $v$ to $S$ such that $|v|$ is as small as possible. To find the sequence of edges whose permutations can be determined, we examine any path from $v$ to $S$ and choose that as our order.

EdgeSequence@k gives a list of the strands in the order that they should be calculated from the generators, which are returned as a list as the first element of the result, of the knot $k$, which is given in modified Dowker notation.
Once we have found the generators, we need to start testing colourings. We first break up $S_m$ into disjoint subsets by cycle lengths. For each of these subsets $T$, we try assigning permutations in $T$ to each of the generators in $v$ in every possible way. There are $|T|^v$ ways of doing so. This exponential is the reason for why we put so much effort into minimizing our set of generators.

Once we set the generator permutations, we generate the permutations for the rest of the edges, and then check that (10) or (11) is satisfied for each crossing. If it is, then the colouring is valid, otherwise, it is not.

We count the total number of valid colourings and our invariant becomes a list of the number of valid colourings of the knot diagram using elements of $S_m$. Each element of the list corresponds to a different subset of $S_m$, where all of the elements in each subset have the same cycle lengths.

After we had applied the Jones polynomial, we found two pairs of knot diagrams that could be equivalent. After setting $m$ to 5 and calculating the number of colourings for each of the four knot diagrams in those two pairs, we found that the two knot diagrams in our of the pairs produced different values, showing that they represent distinct knots. The two knots in the other pair produced the same list for the number of ways in which they could be coloured using elements of $S_5$. Thus, we have at least 250 distinct knot diagrams and potentially as many as 251.

```
ValidColouring[k, e, s, g] :=
ValidColouring[k, e, s, g] =
Block[{v, n = Length@k}, v = Array[0 &, 2 n];
  v[e] = g;
  And @@ Table[PermutationConjugation[v[c[1]]],
    SortBy[k, Length[c[1]]] - 1]
  // If[Mod[n[3] + 1, 2 n, 1],
    v[c[2]],
    InversePermutation[v[c[2]]]] &
  // If[v[c[3]] === 0, v[c[3]] = #;]
  True,
  (*Check if derived value matches
   previously assigned value.*)
  v[c[3]] = #] &,
  {c, s}];
ValidColouring[
  (X, 1, 2, 3, 4), (X, 3, 5, 6, 4, 1), (X, 5, 2, 6, 3, 1), (1, 2),
  (1, 2, 4), (2, 1, 4), (1, 4, 2), ((1, 2, 3), (1, 2, 3))]
```

We now try to see if the knot diagrams which have produced the same result for each of our invariants are equivalent. To show that the two knot diagrams in the pair are equivalent, we try adding a positive kink into the knot diagrams in each of the 10 possible ways. To add a kink, we insert a $k$ into the $k$th position in the MD code and add 1 to all of the other values in the MD code that are greater than or equal to $k$. 

```
ValidColouring[k] :=
ValidColouring[k] =
Block[{e = EdgeSequence[k, s, g],
    w, s = SortBy[If[Order[position[Join@@e, #]]],
      Position[Join@@e, #]] = 1,
    h, Reverse@h] &
  {w} /. {Max@h -> Min@h} &
  Max@Table[position[Join@@e, h]] |.
  Sort] &
  // GroupBy[Permutations@Range@m, h] &
  // Values],
  g, Tuples[p, Length@e]]]];
```
ReidemeisterOne@k gives all the knots that can be obtained by adding a positive kink at an even index to the knot $k$, which is given in modified Dowker notation.

\[
\text{ReidemeisterOne@k} = \text{ReidemeisterOne@k}_\text{MD} :: \text{ReidemeisterOne@k} = \text{Table}[\text{EDT} @
\left\{\left[\text{Map}\left[H + \text{If}[\text{Abs}@H < 2 i, 2 \text{Sign}@H, 0] \&, \text{List}@\text{Build}^k, (2)\right] \cup \left[(2 i, 2 i + 1)\right]\right]\right\}
\]
\]

\[
\text{ReidemeisterThree@k} = \left\{\text{ReidemeisterOne@MD}[3, -5, -9, 7, -1, -8, 10, 4, -2, 6],
\text{MD}[1, -9, -6, 10, -8, -2, 4, 11, -5, 3, 7],
\text{MD}[1, -8, 10, -2, -9, 3, -11, -5, 7, 4],
\text{MD}[1, -8, 10, 7, -11, 9, 3, -5, -2, 6, -4],
\text{MD}[1, -5, -9, 11, 8, 2, 10, 4, -6, -3, 7],
\text{MD}[1, 5, -9, 7, 11, -3, -10, 4, -2, -6, 8],
\text{MD}[1, 8, -10, -7, 11, -9, -3, 5, 2, -6, 4]\right\}
\]

Our two new 11-crossing knot diagrams are found to be equivalent under repeated application of the third Reidemeister move (see Fig. 1).

ReidemeisterThree@k gives all the knots that can be obtained by applying one third Reidemeister move to the knot $k$, which is given in modified Dowker notation.

\[
\text{ReidemeisterThree@k} = \left\{\text{ReidemeisterOne@MD}[3, -5, -9, 7, -1, -8, 10, 4, -2, 6],
\text{MD}[1, -9, -6, 10, -8, -2, 4, 11, -5, 3, 7],
\text{MD}[1, -8, 10, -2, -9, 3, -11, -5, 7, 4],
\text{MD}[1, -8, 10, 7, -11, 9, 3, -5, -2, 6, -4],
\text{MD}[1, -5, -9, 11, 8, 2, 10, 4, -6, -3, 7],
\text{MD}[1, 5, -9, 7, 11, -3, -10, 4, -2, -6, 8],
\text{MD}[1, 8, -10, -7, 11, -9, -3, 5, 2, -6, 4]\right\}
\]

Thus, the two knot diagrams that were producing the same values for each of our in-variants are equivalent. This proves conclusively that we have no fewer and no more than 250 knots with up to 10 crossings.

\[
\text{RolfsenTable@n} = \text{RolfsenTable@n}_\text{nMD} :: \text{RolfsenTable@n} = \left\{\left[\text{MD}[2, 4, 5, 6, 1, 7, 3],
\text{MD}[2, 4, 6, 1, 7, 3, 5],
\text{MD}[2, 5, 6, 7, 1, 4, 3],
\text{MD}[2, 5, 7, 6, 1, 4, 3],
\text{MD}[3, 5, 6, 7, 1, 2, 4],
\text{MD}[3, 5, 6, 7, 2, 1, 4],\right]\right\}
\]

X Knot Graphs

In our calculation of the Rolfsen table, we mainly used three moves, the 2–pass, the third Reidemeister move, and the flype (see Fig. 1). We generated a graph of connections to find equivalent knot diagrams. Now, we run our algorithms with a different set of knot diagrams. We replace distinct prime alternating knot diagrams with all prime alternating knot diagrams. The difference between the sets is that the latter has diagrams that are equivalent with respect to flyping.

From these knot diagrams, all of their non-alternating knot diagrams are generated, each knot diagram is mapped to a vertex, these vertices are connected with edges representing 2–passes, third Reidemeister moves, and flypes (see Fig. 1), and lastly all connected components that were found to be reducible are removed.

CreateGraph[n] gives a graph with minimal irreducible knot diagrams with n crossings as vertices and edges connecting each pair of knot diagrams that are equivalent under one 2–pass, flype, or third Reidemeister move.

CreateGraph[] := Join @@ Array[CreateGraph, 11, 0];
CreateGraph[0] := {{MD, MD, "N/A"}};
CreateGraph[n___Integer] := CreateGraph[n] =
Block[{r, y = Join[Reverse@KnotSort@#, #2] & /@ (KnotSort@# & /@ CandidateKnots@n) \[Union] Flatten[Table[{(k, #, "Reidemeister 3") & /@ ReidemeisterThree@k, {k, #, "2-Pass"} & /@ TwoPass@k}, {k, ValidKnots@n}, 2] \[Union] {}],
{MD[2, 4, 5, 1, 6, 3], MD[2, 4, 5, 1, 6, 3], Flype},
{MD[2, 4, 5, 6, 1, 3], MD[2, 4, 5, 6, 1, 3], Flype},
{MD[2, 4, 6, 5, 1, 3], MD[2, 4, 6, 5, 1, 3], Flype}]
CreateGraph@6
{MD[2, 4, 5, 1, 6, 3], MD[2, 4, 5, 1, 6, 3], Flype},
{MD[2, 4, 5, 6, 1, 3], MD[2, 4, 5, 6, 1, 3], Flype},
{MD[2, 4, 6, 5, 1, 3], MD[2, 4, 6, 5, 1, 3], Flype}]

The result is the full graph of irreducible knot diagrams and their connections. This graph can be used for testing knot invariants. Each invariant must produce the same result for each vertex in every connected component. Below is this graph, but with no labels.

This graph is also available online at http://tiny.cc/RolfsenTableGraph.

XI Utility Functions

Here we include all the functions that are not mathematically interesting, but merely serve as helper functions for those that are. They are included here for completeness and in alphabetical order for ease of access.

All of this code is also available online at http://tiny.cc/RolfsenTableCode.

Build@k gives the extended Dowker notation of the knot k, which is given in modified Dowker notation.

Build@k_MD :=
Build@k = ED[2 Range@Length@k - 1, 2 List@k];
Compactify@k gives the modified Dowker notation of the knot k, which is given in extended Dowker notation.

Compactify@k_ED :=
Compactify@k = MD @@ ((1, Sign[#1] #2))
Mod[Abs@If[OddQ@#1], #1, #2] & /@ # & /@ {List@k'}
// Sort'\[2]
Convert@k gives the minimal modified Dowker notation of the knot k, which is given in extended Dowker notation.

Convert@k_ED := Convert@k = Minimal@Compactify@k;
Data[n] gives the file that graphs of knots with n crossings will be saved to and loaded from.
Data[] gives the file that the graph of all knots with up to 10 crossings will be saved to and loaded from.

\[
\text{Data[n]} \equiv \text{FileNameJoin[\{NotebookDirectory[], "math", "all.m\}];} \\
\text{Data[]} \equiv \text{FileNameJoin[\{NotebookDirectory[], "math", ToString[n <> ".m\}]}; \\
\text{DrawGraph@n displays the graph of knot diagrams with n crossings that it loaded from the appropriate file.} \\
\text{DrawGraph[] displays the graph of knot diagrams with up to 10 crossings.} \\
\text{DrawGraph[n]} \equiv \text{GraphPlot[#1 \rightarrow #2, #3] \&/@ (Data[n]),} \\
\text{EdgeLabeling -> False,} \\
\text{EdgeRenderingFunction ->} \\
\text{Switch[H3, "2-Pass", Red,} \\
\text{"Reidemeister 3", Green, "Flype", Blue, "N/A",} \\
\text{Transparent],} \\
\text{Arrowheads@0, Arrow[H1]] \&],} \\
\text{VertexRenderingFunction ->} \\
\text{(Switch[D3, 1, 1/3,} \\
\text{2 Length[k] \& \& Length[k] < \&\& Total[v == a] \&\& l] \&\& l, 1, 1)] \&; \\
\text{GraphSort@[a,b] returns True if the lists a and b, containing a pair of vertices and an edge name, are in sorted order and False otherwise.} \\
\text{GraphSort[a>List, b>List] := GraphSort[a, b] =} \\
\text{If[a[[1]] === b[[1]],} \\
\text{If[a[[2]] === b[[2]],} \\
\text{Order[a[[3]], b[[3]]] \&\& 0,} \\
\text{SortedQ[a[[2]], b[[2]]],} \\
\text{SortedQ[a[[1]], b[[1]]]];} \\
\text{KnotAssociation@n gives an association that returns the list of all valid knots with n crossings whose alternating form is equal to the given knot.} \\
\text{KnotAssociation@Integer := KnotAssociation@n =} \\
\text{Table[k \rightarrow Select[ValidKnots@n, Abs/@H = k \&],} \\
\text{k, CandidateKnots@n]} \} // Association; \\
\text{MakeGraph@n creates the graph of knot diagrams with n crossings and then saves it to the appropriate file.} \\
\text{MakeGraph[] creates the graph of knot diagrams with up to 10 crossings and then saves it to the appropriate file.} \\
\text{MakeGraph[n] :=} \\
\text{(List @@ #1, List @@ #2, #3) \&@@ CreateGraph@n >> Data@n);} \\
\text{SortedQ@l gives True if the list of knots l is sorted and False otherwise.} \\
\text{SortedQ@l_List := SortedQ@l = 1 \&\& KnotSort@l;
Andrey Boris Khesin

This list can be found online at [http://tiny.cc/RolfsenTable]
XIII ACKNOWLEDGEMENTS

The author is grateful to Prof. Dror Bar-Natan for his neverending support and assistance with this project. Additionally, Prof. Bar-Natan originally suggested the topic of this project.

XIV REFERENCES

[1] Hoste, Jim, Morwen Thistlethwaite, and Jeff Weeks. “The First 1,701,936 Knots.” The Mathematical Intelligencer 20.4 (1998): 33-48. Web.

[2] Kauffman, Louis H. “New Invariants in the Theory of Knots.” The American Mathematical Monthly 95.3 (1988): 195. Web.

[3] Menasco, William W., and Morwen B. Thistlethwaite. “The Tait Flyping Conjecture.” Bulletin of the American Mathematical Society 25.2 (1991): 403-13. Web.