Proceedings of GREAT Day

Volume 2019 Article 13

2020

Space-Efficient Knot Mosaics of Size 7

Gregory Vinal
SUNY Geneseo

Space-Efficient Knot Mosaics of Size 7

Gregory Vinal

sponsored by Aaron Heap, PhD & Doug Baldwin, PhD

Abstract

Mosaic knot theory is a young and exciting area of mathematics. Since it is still in its mathematical infancy, there are many basic questions still unanswered. The main goal of the field is to represent knots on a square grid using a certain set of tiles. From those tiles we can build knot projections. The smallest $n \times n$ grid that each knot with crossing number 10 or less can fit on is unknown. In this paper, we explain how we developed a program that creates and identifies every knot on a 7×7 mosaic, and we find every knot with crossing number 10 or less that can fit on a 7-mosaic with tile number 27 or 29.

Introduction

Mosaic knot theory is an area of knot theory that aims to represent a knot projection on some $n \times n$ grid, using a specific set of tiles. In other words, we aim to restrict knot projections into being composed of the tiles which are shown in Figure 1 laid in a $n \times n$ grid, in order to introduce more structure into otherwise unstructured projection. There are some natural advantages to this approach; namely, given a mosaic representation of a knot projection, that projection can be recreated exactly with minimal information. Additionally, this approach allows computers to analyze a knot projection, since there is a natural representation of a knot mosaic as a matrix.

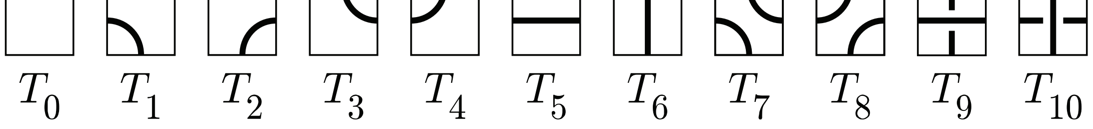


Figure 1: All Mosaic Tiles

It is known that any knot with finite crossings can be represented on a suitably large $n \times n$ grid (n-mosaic)(Lomonaco & Kauffman, 2008). One example of this is shown for the canonical representation of the trefoil in Figure 2. Notice that all the crossings are preserved, and the mosaic on the right is indeed the same projection as the freely-drawn projection on the left. In this way, we have taken a projection that is unregulated and can be arbitrarily complex and converted it into a mosaic that is structured. The field of mosaic knot theory is very new, with the first paper on the subject being published in 2008 (Lomonaco & Kauffman, 2008). Because of this, we know certain things, but other simple questions remain open. For example, we know that the trefoil can be represented on a 4 × 4 grid (and no smaller) and that the figure-8 knot can fit on a 5 × 5 grid minimally. However this property for knots with more than 8 crossings is unknown in general. For each knot with crossing number 10 or less, we set out to find the smallest $n \times n$ grid on which that knot can be represented, and its densest representation, thus completing the table of knots up to and including 10.

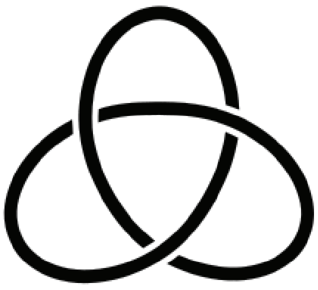
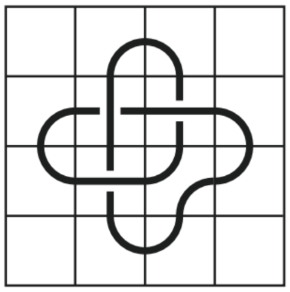



Figure 2: Representing a Projection on a Mosaic

Preliminary Definitions

First, we should offer rigorous definitions for the objects that we will be talking about:

Definition 2.1

A **knot** is a simple closed curve in S^3. A **link** is the nonempty union of a finite number of knots.

Definition 2.2

A **connected sum** of two knots is a knot formed by deleting a segment (1-ball) inside each knot and gluing together the resulting boundary points (1-spheres).

Definition 2.3

A **composite knot** is a knot that is the connected sum of two non-trivial knots. A **prime knot** is any knot that is not composite.

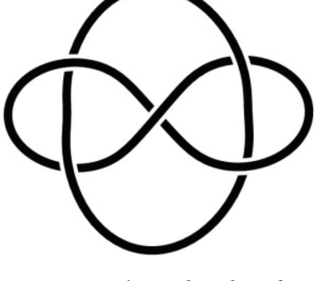


Figure 3: The Whitehead Link

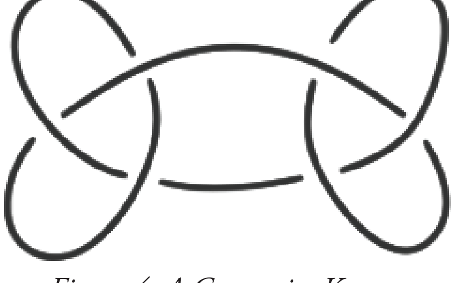


Figure 4: A Composite Knot

It is hard to draw in S^3, so we draw knots on paper. Consider the following definition:

Definition 2.4

A **knot projection** or **knot diagram** is a projection of a knot into a plane with arcs at crossings differentiated into over-crossings and under-crossings. A **link diagram** is a similar projection of a link. In simpler terms, a link is more than one knot in space. Figure 3 shows two unkots that are linked non-trivially. However, Figure 3 and Figure 4 viewed together is also a link, as it is three knots in the same projection. Notice that links can be trivially linked together.

In order to form a connected sum of two knots, we cut each knot at some point and glue the knots together at their boundaries. Figure 4 shows a composite knot. Notice that this knot is the connect sum of two trefoils. Knot theory is interested in distinguishing knots, and so we need to define equivalence between knots:

Definition 2.5

Two knots, K_1,K_2 are said to be **knot equivalent** if there is a continuous function, i: $[0,1] \times S^3 \rightarrow S^3$, such that

- $i(0, S^3)$ is the identity function
- $i(1, K_1) = K_2$
- For all $t \in [0,1]$, $i(t,S^3)$ is a homeomorphism to S^3.

We call i an ambient isotopy. We also need to define how we classify knots:

Definition 2.6

The fewest numbers of crossings needed to represent a knot is called its **crossing number**.

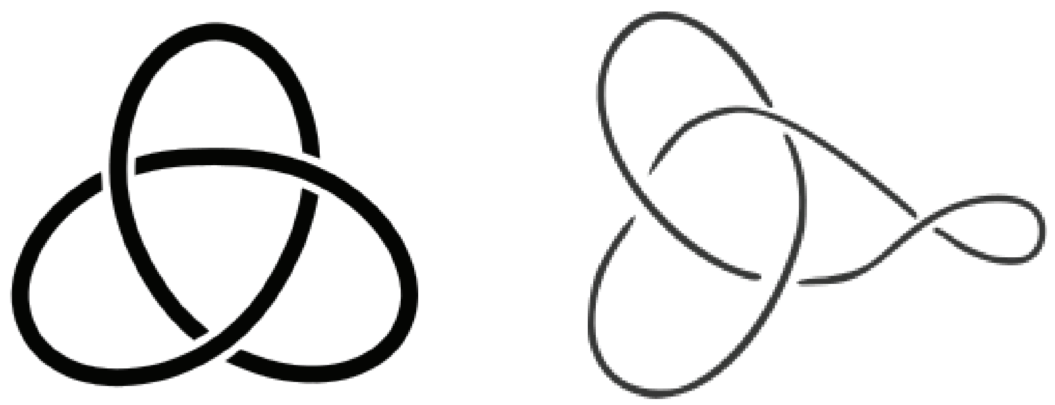


Figure 5: Two Projections of the Trefoil

The two knots shown in Figure 5 are the same, and the continuous deformation (ambient isotopy) from one to the other is pretty clear. However, these are two different knot projections. We can quickly tell because the knot projection on the left has three crossings and the knot projection on the right has 4 crossings. When we say that two knots are the same, we are saying that the two knots differ only in ways like the differences shown in Figure 5, namely that there is a smooth, continuous deformation from one into the other, which can be rigorously defined using an ambient isotopy. Notice that although the knot projection on the right has 4 crossings, the knot it represents has crossing number 3, since 3 is the minimal amount of crossings in any projection of the knot. We must also define what we mean by a knot mosaic. The following three definitions will suffice:

Definition 2.7

A **connection point** of a tile is a midpoint of a tile edge that is also the endpoint of a curve drawn on the tile.

Definition 2.8

A tile is **suitably connected** if each of its connection points touches a connection point of an adjacent tile.

Definition 2.9

An $n \times n$ **knot mosaic**, or **n-mosaic**, is an $n \times n$ matrix whose entries are suitably connected mosaic tiles. Consider the following examples:

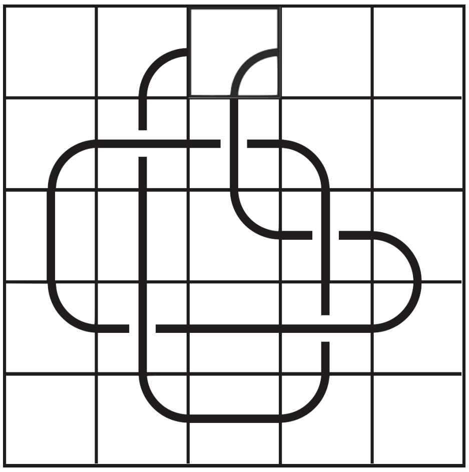


Figure 6: This is not Suitably Connected

The matrix shown in Figure 6 is not a knot mosaic. Notice how in Figure 6, there is a tile in the (1,3) position that is not suitably connected. In particular, there is a connection point on the right edge that is not touching a connection point of the tile adjacent to it. Therefore, this matrix has a tile that is not suitably connected and is therefore not a knot mosaic. However, the knots shown in Figure 7 are both knot mosaics because every tile has connection points that are adjacent to connection points on the tiles that are adjacent to them.

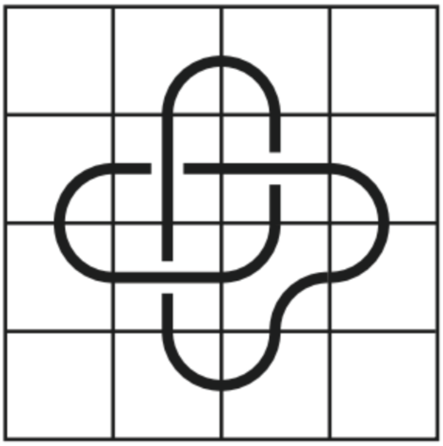
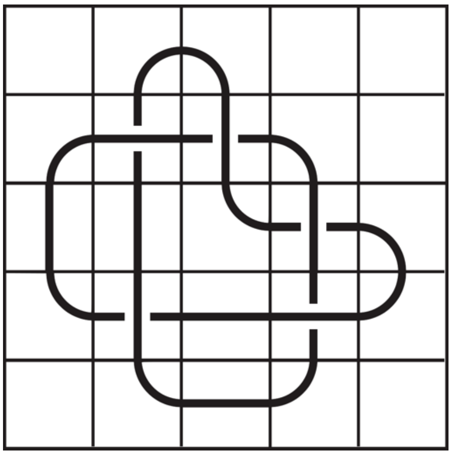



Figure 7: Two Knot Mosaics

There are some characteristics of a knot mosaic that we would like to optimize, arising from natural questions that one might ask. What is the smallest size mosaic on which a specific knot can fit? What is the smallest number of non-blank tiles needed to create a specific knot? Consider the following definitions.

Definition 2.10

The **mosaic number** of a knot is the smallest $n \times n$ grid on which that knot can fit.

Definition 2.11

We define the **tile number** of a knot as the fewest number of non-blank tiles that are needed to represent that knot.

Definition 2.12

We define the **minimal mosaic tile number** of a knot to be the fewest number of tiles needed to represent a knot on its minimal mosaic.

Minimizing the number of tiles that must be used is not as straight forward as one might think; it is not always the case that tile number can be realized on a minimal mosaic. More specifically, the 9_{10} knot can fit on a 6×6 grid using 32 tiles minimally, but it can fit on a 7×7 grid using only 27 tiles (Heap & Knowles, 2019). Figure 8 shows the 9_{10} knot on a 6-mosaic using 32 tiles. This is the fewest number of tiles that are needed to represent this knot on a 6-mosaic (Heap & Knowles, 2019). However, we can see in Figure 9 that the 9_{10} knot can be represented using only 27 tiles. Therefore on the smallest mosaic, the fewest number of tiles needed to represent the 9_{10} knot is 32. However, the fewest number of tiles needed to represent the 9_{10} knot over any mosaic size is 27 (Heap & Knowles, 2019). Therefore, the minimal mosaic tile number of the 9_{10} knot is 32, but the tile number of this knot is 27.

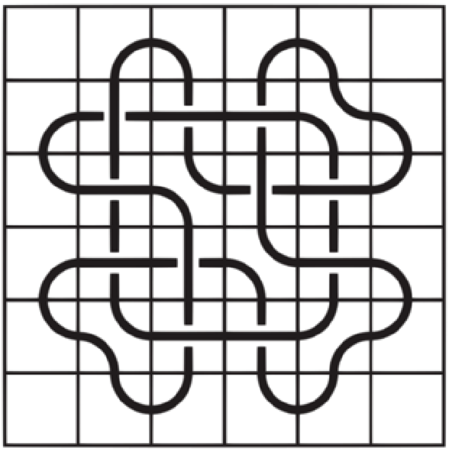


Figure 8: Space-Efficient Representation of 9_{10} with Mosaic Number Realized

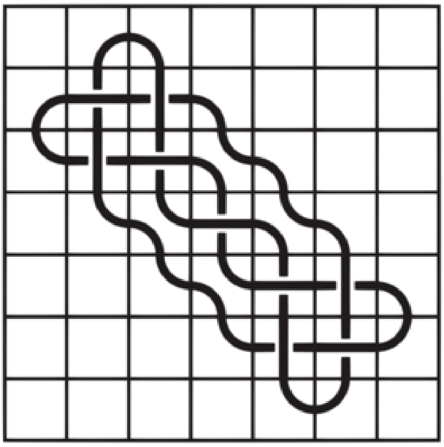


Figure 9: 9_{10} with Tile Number Realized

Definition 2.13

A **space-efficient *n*-mosaic** is an *n*-mosaic in which the number of tiles used is minimized for a mosaic of size *n*.

Definition 2.14

A knot mosaic is **minimally space-efficient** if it is a space-efficient mosaic and the mosaic number is realized. Notice that both representations of the 9_{10} knot shown in Figures 8 and 9 are space-efficient mosaics. However, Figure 8 is a minimal space-efficient mosaic, since it is space efficient and the mosaic number is realized. With these new terms in hand, we can rigorously define the goals of our research; we hope to find the mosaic number, minimal mosaic number, and space-efficient representation of every knot with crossing number 10 or less.

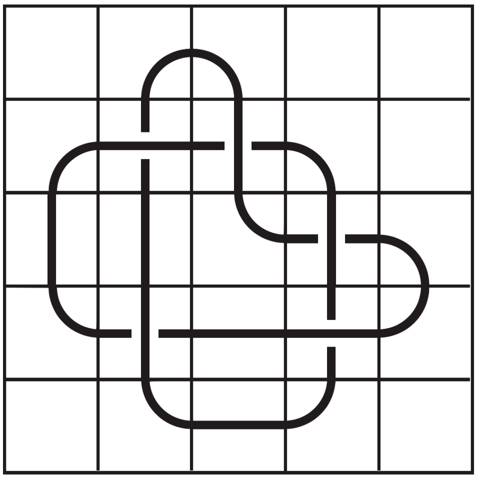


Figure 10: 5_1 with Mosaic Number Realized

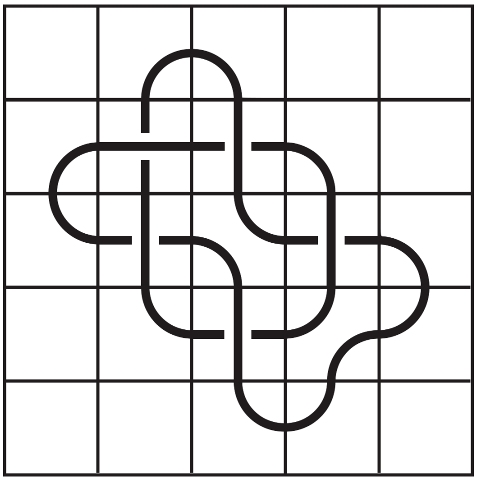


Figure 11: Space-efficient representation of 5_1

Space-efficient 7-mosaics

The concept of tile number and space-efficiency were introduced by Heap & Knowles (2019). There, he provided all possible space-efficient layouts for an *n*-mosaic for all $n \leq 6$. Heap has also already published an exhaustive list of all knots with mosaic number less than or equal to six (Heap & Knowles, 2018). The goal of our research is to complete the table of knots for crossing number up to and including 10 by using only a 7-mosaic. In other words, we hope that every knot with crossing number less than 10 can be realized on a 7-mosaic or smaller, and we are looking to find the space efficient representations of each of these knots. It is known that all knots with crossing number 8 or less can fit on a 6-mosaic or smaller Heap & Knowles, 2018). Therefore, when we say that we are looking to find all knots with crossing number 10 or less, we are really only looking for the remaining 9 & 10 crossing knots that were not able to be realized on a 6-mosaic. Heap and LaCourt had previously developed a list of all space-efficient layouts of 7-mosaics. They also showed that if a knot is in its space-efficient layout, the interior of the knot mosaic will not contain a T_5 or T_6 tile (as shown in Figure 1). In other words, the only tiles needed to make a space-efficient knot mosaic are the T_1, T_2, T_3, & T_4 tiles to build the outer shell, and the T_7, T_8, T_9, & T_{10} tiles are used to fill the interior of the layout, since those have the densest possible configuration and four connection points per tile.

Theorem 3.1

If a 7-mosaic is in a space-efficient layout, then it fills one of the layouts shown in Figure 12 (Heap & LaCourt, 2019):

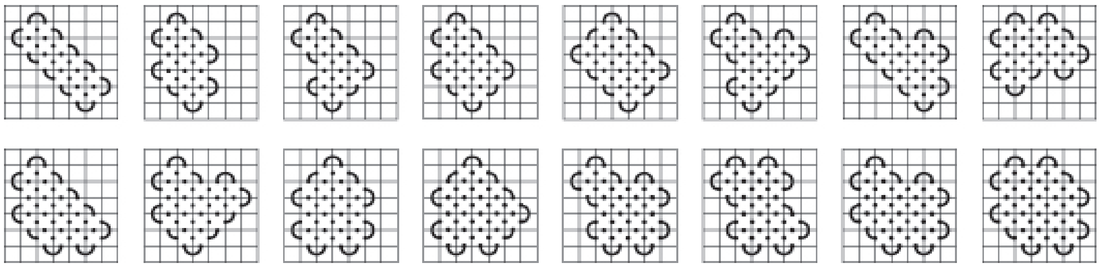


Figure 12: All Space-efficient 7-mosaics

We then set out to build a process for finding all knots with crossing number 9 or 10 that have mosaic number 7, and their corresponding space-efficient representation. Additionally, we also tried to find all knots with mosaic number 6 and minimal mosaic tile number 32 whose tile number is only realized on a 7-mosaic, such as the 9_{10} knot described above. We ran the algorithm for each of the space-efficient 7-mosaic layouts. At the time of writing this, the first 4 layouts of the 7 × 7 mosaic shown in Figure 12 are completed, and the program is still running on the larger layouts. We have found several new 9 & 10 crossing knots, and the results so far are shown below:

Theorem 3.2

The following prime knots have mosaic number 7 and tile number 27:

1. 9_6, 9_{15}, 9_{18}

2. 10_5, 10_6, 10_7, 10_8, 10_9, 10_{10}, 10_{13}, 10_{14}, 10_{15}, 10_{16} , 10_{17}, 10_{18}, 10_{19}, 10_{24}, 10_{25}, 10_{26}, 10_{29}, 10_{30}, 10_{31}, 10_{32}, 10_{33}, 10_{35}, 10_{36}, 10_{38}, 10_{39}

Theorem 3.3

The following prime knots have mosaic number 7 and tile number 29:

1. 9_{22}, 9_{25}, 9_{29}, 9_{30}, 9_{32}, 9_{33}, 9_{34}, 9_{36}, 9_{38}, 9_{39}, 9_{42}, 9_{43}, 9_{44}, 9_{45}, 9_{47}, 9_{49}

2. 10_{23}, 10_{27}, 10_{37}, 10_{40}, 10_{42}, 10_{43}, 10_{45} *through* 10_{57}, 10_{67} *through* 10_{73} , 10_{79}, 10_{82}, 10_{83}, 10_{84}, 10_{86}, 10_{87}, 10_{90} through 10_{95}, 10_{101}, 10_{102}, 10_{103}, 10_{106}, 10_{107}, 10_{112},

10_{113}, 10_{114}, 10_{117}, 10_{128} *through* 10_{136}, 10_{145}, 10_{146}, 10_{147}, 10_{149} *through* 10_{153}, 10_{156}, 10_{158}, 10_{160}, 10_{161}, 10_{162} , 10_{163} , 10_{164}

In other words, we have shown that all but 2 (so far) knots with crossing number 9 and all but 29 of the knots with crossing number 10 have mosaic number 7 or less.

We also found all knots with crossing number up to 32 that share the 9_{10} property of tile number not being realized on a minimal mosaic. As a matter of fact, Heap found thirteen 9- and 10-crossing knots that had minimal mosaic tile number 32 and mosaic number 6. We have found all 13 on a 7-mosaic using fewer tiles (either 27 or 29).

Theorem 3.4

The following prime knots have mosaic number 6, minimal mosaic tile number 32, and tile number 27:

1. 9_{10}

2. 10_{11}, 10_{20}, 10_{21}

Theorem 3.5

The following prime knots have mosaic number 6, minimal mosaic tile number 32, and tile number 29:

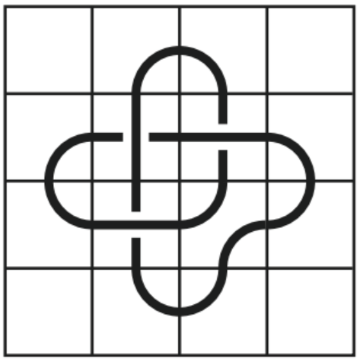
1. 9_{16}, 9_{35}

2. 10_{61}, 10_{62}, 10_{64}, 10_{74}, 10_{76}, 10_{77}, 10_{139}

Of course, the proof of this theorem is a series of pictures, and the validity of these theorems is not dependant completely on the validity of the algorithm.

Our Method of Proof

The smallest layout for the 7-mosaic has 13 entries that must be filled, each populated with one of four possible tiles. Naïvely, this results in 4^{13} possible options, or 67,108,864 possible knots and links. For the largest layout, which has 23 entries that must be filled, there are 4^{23}=70,368,744,177,664 possible configurations. Over all possible layouts there are about 77 trillion possible configurations. Therefore, we employed computers. In order to do this, we had to develop a program that would produce and analyze all knots that can be placed on a 7-mosaic. James Canning and I, with the aide of Aaron Heap and Doug Baldwin developed a program that, when given a layout for any *n*-mosaic, would fill that layout with every possible knot and link that can fit in it, and identify every prime knot that among that list. The first step is to utilize the natural translation from a knot mosaic to a matrix. In other words, we can represent a knot mosaic as a matrix, which is very easy for a computer to produce, store, and work with. We simply replace every tile T_i and replace it with the number *i*. This correspondence is shown in Figure 13.



$$\begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 10 & 9 & 1 \\ 3 & 9 & 8 & 4 \\ 0 & 3 & 4 & 0 \end{pmatrix}$$

Figure 13: Representing a Mosaic as a Matrix

We also need a small Lemma:

Lemma 4.1

Given a knot mosaic matrix, the first non-zero tile must be T_2, when reading across rows from left to right.

Proof: Let M be a knot mosaic matrix. Then, for the base case assume that the first non-zero tile is in the (1,1) position. Then, it cannot have any connection points on its left edge. Additionally, it cannot have any connection points on it right edge. Additionally, all tiles have at least 2 connection points. Therefore, if a tile is in the (1,1) position and is non-zero, it must have connection points only on its right and bottom edges. Therefore it must be T_2. Now, assume that the first non-zero tile is in the $(1,j)$ position. In other words, assume $M(1,\ell) = 0$ for all $\ell < j$. Then, for the same reasons as above, the connection points of this tile must be on the right and bottom edges, and must therefore be T_2. Lastly, assume that the first non-zero tile is in the (i,j) position. In other words, assume $M(i, \ell) = 0$ for all $\ell < j$ and the first i-1 rows are all zeros. Then, for the same reasons as above, the connection points of this tile must be on the right and bottom edges, and must therefore be T_2. Therefore, no matter where the first non-zero tile occurs, it must be T_2, as desired.

We now need to define an integral part of our research, and that is Dowker-Thistlethwaite (DT) notation. In 1983, M. Thistlethwaite and C. Dowker published a paper that provided a way to classify any knot projection. Moreover, they provided an algorithm that can take a knot projection with n crossings and represent it as a sequence of n even integers. They also proved that given a DT notation, a knot projection that produced the notation can be recovered (Dowker & Thistlethwaite, 1983). The algorithm is as follows:

1. First, pick a starting point and orientation of the knot.
2. Follow the path of the knot, and when a crossing is encountered, denote that crossing with a 1. Denote the next crossing encountered with a 2, and so on.

3. If a crossing is an even number, *n*, and the strand is an under-crossing, denote it as -*n*.

4. Every crossing will have one even and one odd number associated to it (Dowker & Thistlethwaite, 1983).

5. Order the odd numbers sequentially.

6. The even numbers in the order that they appear (when paired with the ordered odd numbers) is the DT notation of the knot projection.

Consider the following calculation of the DT notation of this projection of the 4_1 or Figure 8 knot:

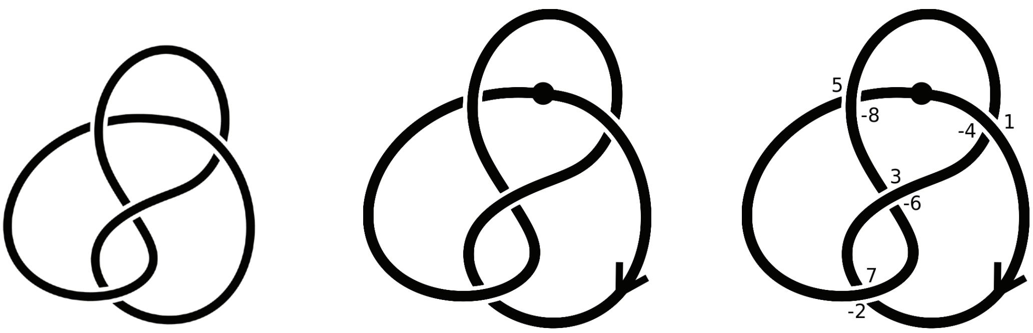


Figure 14: Finding the DT Notation of the 4_1 knot

Then, list in increasing order the odd numbers and match their corresponding even numbers:

$$\begin{bmatrix} 1 & 3 & 5 & 7 \\ -4 & -6 & -8 & -2 \end{bmatrix}$$

This leaves the final DT notation of [-4,-6,-8,-2] for this projection of the 4_1 knot, with the given starting point and orientation. There are a few things that cannot be assumed. First, the DT notation of a knot projection is tied to the projection of a knot, not the knot itself. As a consequence, a knot projection with *n* crossings will have a DT notation of length *n*, regardless of the crossing number of the knot depicted. Consider, once again, the knots originally shown in Figure 5 shown in Figure 15. These are two different projections of the same knot. However, the projection on the left produces the DT notation [4,6,2] (when the starting point is at the highest point and going counterclockwise), and the projection on the right yields the DT notation [4,8,6,2] (with the same starting condition). For any projection of a knot with crossing number *n*, a DT notation of that projection is reduced if and only if it is of length *n*. In this case, the DT notation produced by the knot projection on the left is reduced and the DT notation obtained by the knot projection on the right is unreduced, as the crossing number of this knot is 3.

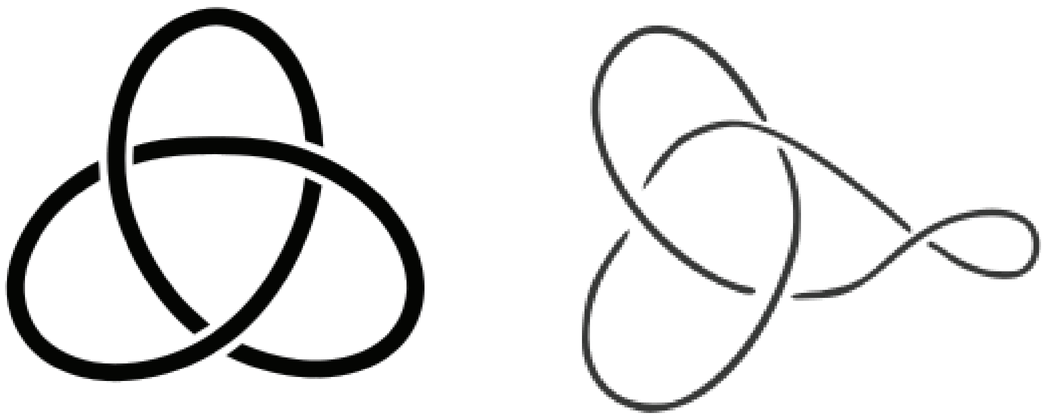


Figure 15: Two Projections of the Trefoil again

Additionally, the same projection can produce many different DT notations, based on the starting position and orientation chosen. Consider once again Figure 14. Based on the starting point and orientation chosen, the DT notation produced was [-4,-6,-8,-2]. However, if we keep the same starting point but have the opposite orientation, we obtain the DT notation [4,6,8,2]. Therefore, knot projections do not produce unique DT codes.[1] However, there are a number of useful attributes that the DT notation has. First, a DT code does uniquely describe a knot projection. In other words, given a DT notation, the knot projection that produced it can be recovered. Additionally, there is an algorithmic way to reduce a given DT notation to its reduced, canonical form. Also, the canonical DT notation uniquely identifies prime knots. In other words, given a prime knot, it has a unique reduced DT notation, down to a factor of -1. Lastly, the main benefit of using DT notation is that computers can compute it, given a way to "read" a knot projection. The second step in our research, then, was to write a code that would receive a matrix representation of a knot mosaic as input and output the DT notation of that matrix.[2] This was done by writing out how to "walk" along a mosaic. Consider the following algorithm:

1. First, starting in the (1,1) position and reading across rows, locate the first non-zero tile, and record the index of that tile, say the (*i,j*) position. We knot from above that this must be T_2. Therefore, the tile in the (*i,j*+1) position is non-zero, as the mosaic is suitably connected.

2. Then, initialize the current tile to be the tile in the (*i,j*+1) position, and the previous tile to be (*i,j*). Additionally, initialize the crossing count to 1.

3. While the index of the current tile is not equal to (*i,j*), do the following:

1 However, both of these DT notations are reduced, as they are both of length 4 and the projection from which they were derived represent a knot with crossing number 4.

2 Or more precisely the DT notation of the projection that the matrix represents

a) If the current tile is a crossing tile (T_9 or T_{10}), record an ordered pair whose first entry is the index of the current tile and whose second entry is the crossing count. Add 1 to the crossing counter.

b) if the crossing is and under-crossing and the crossing count is an even, multiply the second entry of the ordered pair by -1.

c) Then, using the current tile type and the relative position of the previous tile, set the previous tile's value to be the index of the current tile, and increment the current tile's row or column value by ±1, depending on the tile type (Example below).

4. Take the list of ordered pairs of indices and crossing count, and order them by the first value.

5. Then, for every 2 ordered pairs that have the same first element, create a new unordered pair of their second elements.

6. Order each of these unordered pairs by placing the odd element first and the even element second.

7. Order the ordered pairs by their first indices.

8. Create a list whose entries are the second indices of each ordered pairs in the order that they appear.

In short, steps 1 and 2 are initialization steps, step 3 is the part of the algorithm that tells the computer how to trace the knot, and steps 4 through 8 are simply formatting the information and distilling the DT notation of the mosaic.

Step 3(c) refers to using the previous tile and current tile to decide which tile to move to next. As promised, consider the following example: suppose we are in some (m,n) position, and the previous tile has coordinates $(m,n\text{-}1)$, or to the left of the current tile. Additionally, assume we know that the tile in the (m,n) position is tile T_8. Then by following the curve of the strand on the tile, we know that the next tile we must move to is the tile directly above the tile we are currently occupying. In other words, we set the previous tile value to be (m,n), and set the current tile value to $(m\text{-}1,n)$ and iterate the while condition again. This is how the computer can walk along the knot, and this information is encoded as a long switch statement, giving every possible situation and instructions for how to handle each case.

Additionally, at this step we sort out the knots and the links, and reject the links, since we are only looking for prime knots. The second part of this function counts how many tiles we visit. We can calculate how many strands are on the mosaic by counting the number of non-blank tiles, and adding to that the number of 7s, 8s, 9s and 10s since these tiles each contain two stands, so they should be counted twice. If we have not traced every strand, i.e. the number of tiles that we visited is less than the number of strands, we know that we have somehow completed a closed loop without traversing every strand on the mosaic, which means that there is some other loop on the mosaic which was not traversed.

Next, we needed a way to get a computer to reduce the Dowker-Thistlethwaite notation produced in the last step. Thankfully, in the 1990s, Morwen Thistlethwaite and others developed a computer program called *Knotscape* that can reduce a DT notation to the canonical, reduced form. Fortunately, this program does not reduce the DT notations of composite knots. This is beneficial because we are only concerned with the mosaic number of prime knots. We took the code for this program and slightly adapted it to fit our needs. More specifically, the program initially read from a file, and produced output to a file. The program also only read a single knot at a time. We changed the program to receive standard input and produce standard output, as well as to run on a loop while there is input still being supplied. Additionally, the program in its unadapted form returns an empty output when presented with the DT notation of a composite knot. So, we recorded the output, and if it was empty, we knew that the knot was a composite knot and we are not concerned with composite knots.

Lastly, we just needed a function that could identify a knot given its unique, reduced DT notation. The canonical DT notations for all knots with crossing number 16 or less is well known and readily available. Therefore, we compiled a table of all knots with crossing number 10 or less, with the entries of each row being the name of the knot (using the Rolfsen naming system), and the reduced DT notation of that knot. Then, we could search the table for the DT notation of that knot, and print the name of the knot.

Hence, the combination of these three functions could take any knot mosaic and produce the name of the depicted knot if it is a prime knot, or denote it as a link or composite knot if that is the case. Now, we needed a way to produce every knot mosaic, and we would then have everything we needed to sort through all possible knots.

We did achieve this, but we made it a little more efficient. The basic goal is filling each layout with the T_7, T_8, T_9, & T_{10} tiles. When we are filling a particular layout, we know that it has a certain number of blank tiles. For ease of reference, suppose it has 13 blank tiles that need to be filled. We could simply generate all possible vectors of length 13 using the 4 possible entries, and systematically populate the layout using each vector. In other words, for each vector produced, we place the first vector component into the first available tile (reading left to right along rows, then top to bottom down the columns, like a book), the second vector component into the second blank tile, and so on until the entire layout is populated. By doing things this way, we simply need to generate every possible vector of length 13 to produce every possible knot and link that can be represented on the given knot mosaic. However, we are not concerned with the vectors that contain all 7s, for example, because that vector, and by extension the knot mosaic it produces, will not contain any crossings so there is not a possibility of it being a non-trivial knot. Therefore, we only produced every vector that had at least 9 crossings to populate the mosaic layouts, as we needed the knot mosaic to have a lest 9 crossings for it to represent a knot with crossing number 9.

The final program worked as follows, using the functions outlined above:

1. Choose a 7-mosaic layout that has *n* blanks.
2. Build every possible vector with 9 or more crossings of length *n*
3. For every vector, produce a 7-mosaic a by filling the 7 layout with the entries of the vector.
4. Find the unreduced DT notation of each of those mosaics, and eliminate links.
5. Reduce each DT notation, and eliminate composite knots.
6. Look up and print the name of the prime knot that corresponds to each reduced DT notation.
7. Repeat this process for every 7-mosaic layout.

Then, the combination of all these programs can produce, analyze, and name every knot that can fit on a 7-mosaic.[3] We can now look at some of the results that we found.

Ongoing Work

In order to verify that the algorithm we created could accurately find all the knots we were looking for, we tested the algorithm using all the 6×6 mosaics from Heap (2018). The results exactly matched the results that Heap found by hand. Therefore, we concluded that the program could accurately find all knots. This of course strongly suggests that our algorithm is sound and produces correct and concise results, but it does not prove it. Thankfully, our final results are not entirely reliant on the accuracy of the algorithm we wrote.

The output of the algorithm is a list of at most 150 knots mosaics. We hope that all knots with crossing number up to 10 can fit on a 7-mosaic. If the algorithm produces some mosaic for every knot with crossing number up to 10, that will suffice to show that every knot with crossing number up to 10 has mosaic number less than or equal to 7. However, this will not necessarily find the minimal mosaic tile number for each of these knots. That fact can be checked by hand once a list of all the knot mosaics has been produced. In this way, we are using the algorithm generated to sort through the high volume of possible knots, but once that number has been reduced to a small number of results, we can verify the results of the algorithm by hand. Because of this,

3 These programs only use the fact that the mosaic is a 7-mosaic in the phase where the layout is being populated. These functions work for any *n*-mosaic, but the reduction program only can identify knots of up to crossing number 16.

the validity of the algorithm, although beneficial, does not impact the validity of our findings.

While the algorithm is running, I have developed a website where a user can build a knot mosaic and have the knot that was drawn identified. This website required a new function that would decide if a given knot mosaic matrix was suitably connected or not.

In short, the function cycles through each entry of the matrix, checks where it has connection points, and checks that the four neighbors of the tile (should they exist) all have connection points where appropriate. If it is decided that the knot mosaic matrix is indeed suitably connected, then the rest of the program as outlined runs, and the knot is identified.

Our hope is that in the near future, the program finds all the remaining 9 and 10 crossing knots. Additionally, we hope that the website will be hosted for the public to use, to further aide in the research of knot mosaics.

References

Dowker, C. H., & Thistlethwaite, M. B. (1983) Classification of knot projections. *Topology and Its Applications*, *16*:19–31. https://doi.org/10.1016/0166-8641(83)90004-4

Heap, A., & Knowles, D. (2018) Tile number and space-efficient knot mosaics. *Journal Knot Theory Ramifications*, *27*. https://doi.org/10.1142/S0218216518500414

Heap, A., & Knowles, D. (2019) Space-efficient knot mosaics for prime knots with mosaic number 6. *Involve, a Journal of Mathematics*, *12*(5), 767-789. https://doi.org/10.2140/involve.2019.12.767

Heap, A., & LaCourt, N. (2019). Space-efficient prime knot 7-mosaics. *arXiv preprint* arXiv:1912.13093.

Lomonaco, S. J., & Kauffman, L. H. (2008). Quantum knots and mosaics. *Quantum Information Processing*, *7*(2-3), 85-115. https://doi.org/10.1007/s11128-008-0076-7