AN INVESTIGATION OF DIAGONAL KNOTS

JACKSON ARNDT, MALIA BEATTY, PAYTON MCBURNEY, AND KATHERINE VANCE

ABSTRACT. In 2003, Ozsváth, Szabó, and Rasmussen introduced the τ invariant for knots, and in 2011, Sarkar published a computational shortcut for the τ invariant of knots that can be represented by diagonal grid diagrams. Previously, the only knots known to have diagonal grid diagram representations were torus knots. We prove that all such knots are positive knots, and we produce an example of a knot with a diagonal grid diagram representation which is not a torus knot.

1. INTRODUCTION

*****Need to write content*****

2. PRELIMINARIES

While diagrams are the standard way to visually represent knots, *grid diagrams* can also be used to specify knots. A grid diagram of size n has n rows and n columns. The rows are numbered bottom to top, while the columns are numbered left to right. In each row and column, there will be exactly one X-marking and one O-marking, meaning there are n X markings and n O markings. In order to recover the knot given by a diagram, connect the X's to the O's in each column. In each row, connect to the O's to the X markings. Every vertical strand should pass over the horizontal strand at a crossing.

Two permutations, called $\sigma_{\mathbb{X}}$ and $\sigma_{\mathbb{O}}$, where $\mathbb{X}$ is the set of X-markings and $\mathbb{O}$ is the set of O-markings, define the appearance of a grid diagram. The permutation $\sigma_{\mathbb{X}}$ contains numbers which denote which row the X marking will be in. The first number in the permutation gives the row in which X will be located in the first column, the second number gives the row in which X will be located in the second column, and so on. The same goes for the O markings. Fig. 1 shows an example of a grid diagram, along with the knot diagram recovered from it.

2.1. Knot Floer Homology and the τ Invariant.

Knot Floer Homology and the τ Invariant. Knot Floer Homology, abbreviated HFK, is a "package" of knot invariants defined in the early twenty-first century by Ozsváth, Szabó, and Rasmussen. These invariants can be defined combinatorially using grid diagrams [MOST07]. We will focus on the τ invariant, an integer-valued invariant which is part of the HFK package. The τ invariant has some nice geometric properties, including the following bound on slice genus:

Theorem 1 (Osváth-Szabó, Sarkar). *For any knot K, $|\tau(K)| \leq g_4(K)$.*

The τ invariant is very difficult to compute. Fortunately, for some specific cases such as in Theorem 2, there is a computational shortcut (although for larger grid diagrams, the computation is still quite lengthy, even using the shortcut). First, we define $\mathcal{J}(A, B)$ as a

Date: June 12, 2019.

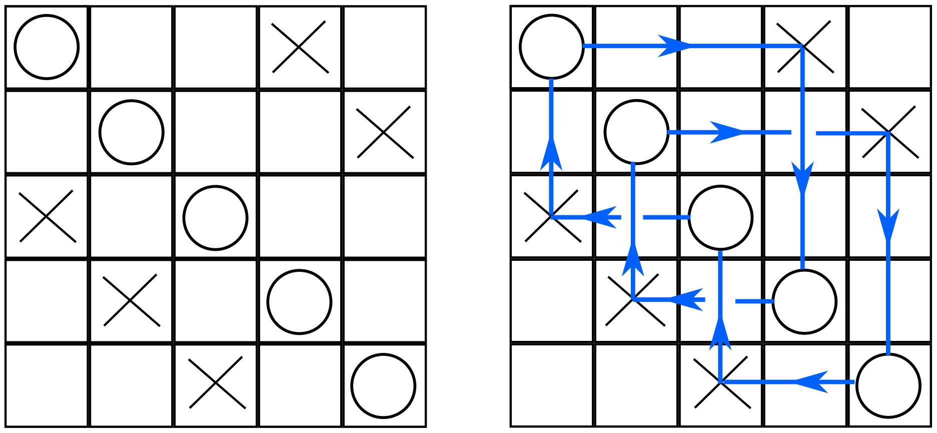


FIGURE 1. A grid diagram representing the trefoil, and the knot diagram recovered.

symmetric, bilinear function on sets in $\mathbb{R}^2$. The function $\mathcal{J}(\{a\}, B)$ is half the number of points in B which are to the top right and bottom left of a, and

$$\mathcal{J}(A, B) = \sum_{a \in A} \mathcal{J}(\{a\}, B).$$

Theorem 2 (Sarkar)**.** *If K has a grid diagram with the O's all on the diagonal from top left to bottom right, then*

$$\tau(K) = \mathcal{J}\left(\mathbf{x} - \frac{1}{2}(\mathbb{X} + \mathbb{O}), (\mathbb{X} - \mathbb{O})\right) - \frac{n-1}{2}.$$

The set $\mathbb{X}$ in this equation is the set of X markings, $\mathbb{O}$ is the set of O markings, $\mathbf{x}$ is the set of points in between the O's on the grid diagram along with one point on the bottom left corner of the grid, as shown in Fig. 2, and n is the size of the grid.

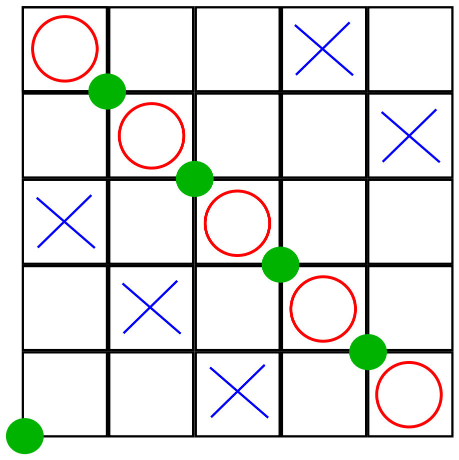


FIGURE 2. This is a color coded grid diagram

2.2. **Diagonal Knots.**

Definition 1. A *diagonal knot* is a knot which can be represented on a grid diagram with all of the O's on the diagonal from top right to bottom left.

Since we know that τ can be easily calculated for diagonal knots, it would be helpful to know if there are specific knot types that can always be represented diagonally. For example, torus knots can always be represented diagonally. In fact, Sarkar used his shortcut on torus knots [Sar11]. However, we would like to know if other knots besides torus knots are diagonal knots.

Proposition 1. The proportion of grid diagrams of size n that are diagonal is $\frac{1}{n!}$.

Proof. We will prove the proposition by examining the process of creating a grid diagram. To begin, we place the O markings. There is only one possible way to place the O markings on the diagonal. Let k be the number of ways to place the X markings given the O permutation. Then k is the total number of diagonal grid diagrams with grid size n.

Now, in order to find the proportion, we compute the total number of grid diagrams with grid size n. We being by placing the O markings in the grid. In the first column, there are n different ways to place the O. In the next column, counting from left to right, there are $n-1$ ways to place the O. In total, there will be $n(n-1)(n-2)...(2)(1) = n!$ ways to choose the position of the O's. Once the O's are in place, we now place the X markings. Because the same number of spaces are taken up by the O's now as when they were placed diagonally, the number of ways to place the X markings is still k. Now, to find the total number of grid diagrams with grid size n:

$$\text{Total number of grid diagrams with grid size } n : k \times n! = (k)(n!).$$

Finally, to compute the proportion of grid diagrams that are diagonal, we can divide the number of possible diagonal grid diagrams by the total number of grid diagrams to give us:

$$\frac{k}{(k)(n!)} = \frac{1}{n!}.$$

□

2.3. **Torus Knots and Links.** Torus links are an infinite family of links. They are called such because they can wrap around a torus without crossing. A torus link, $T_{p,q}$ where $p, q \neq 0$, is formed by taking p horizontal strands. If q is positive, the bottom q strands are wrapped up and over the remaining strands, beginning from the bottom. If q is negative, the q strands are wrapped down and over the remaining strands, starting with the top. If $|q| > |p|$, then the process restarts, beginning with the first strand that was pulled over the top. If $p < 0$, the diagram will not change; however, the orientation in which the link wraps around the torus will be opposite. Once the overlapping is complete, the strands are connected in a fairly simple manner. The strand that ends at the top will loop above and connect with the initial top strand. The strand that ends second to the top will loop above and connect with the strand that began second from the top, and so on. The resulting knot or link will look like the diagram in Fig. 3.

While some torus links are knots, because they only have one component, others have multiple components. In our investigation, we consider only knots rather than links.

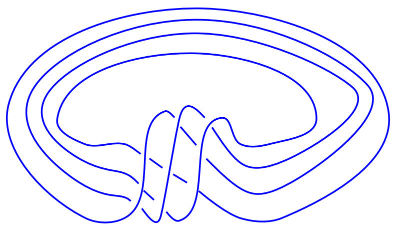
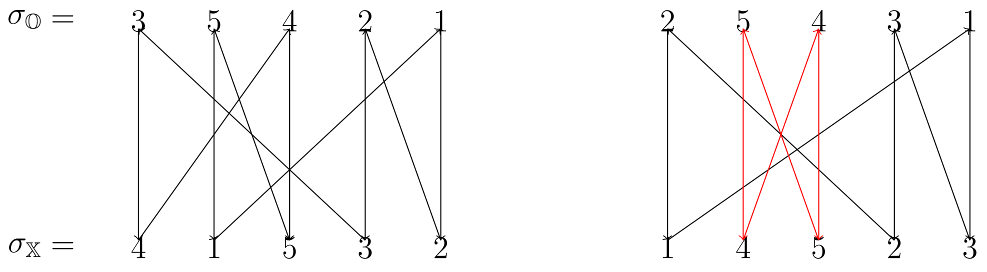


FIGURE 3. A torus knot, $T_{4,3}$.

3. The τ invariant and diagonal knots

3.1. **Methods.** Working with knots can be an arduous task for the human brain, and computers can be helpful.

3.1.1. *Testing for a Knot.* Since our ultimate goal is working with the τ invariant, we are only interested in knots. To this end, we had to think of a way to look at a grid diagram and determine if it was a knot or a link. We noticed that if you arrange the permutations of a grid diagram so the $\sigma_{\mathbb{O}}$ permutation is right above the $\sigma_{\mathbb{X}}$ permutation (the order does not matter), you can draw arrows alternating between the two permutations. Draw the arrows by going from the number in $\sigma_{\mathbb{O}}$ to the one directly below it in σ_X. Then draw an arrow from the $\sigma_{\mathbb{X}}$ number to the equivalent number in $\sigma_{\mathbb{O}}$ and so on. If the diagram is that of a knot, the arrows will loop endlessly between all numbers in the two permutations. If it is a link, then more than one loops of arrows of will be necessary to connect all the numbers.



$\sigma_{\mathbb{O}} =$ 3 5 4 2 1
$\sigma_{\mathbb{X}} =$ 4 1 5 3 2
2 5 4 3 1
1 4 5 2 3

In order to make a computer do this, we wrote a program that takes in the $\sigma_{\mathbb{O}}$ and $\sigma_{\mathbb{X}}$ permutations of a grid diagram and determines whether it is a knot or a link. The function takes two arrays as parameters, $\sigma_{\mathbb{O}}$ of the knot and its σ_X. It first checks to make sure both arrays are the same length, terminating the program if they are not. Second, the program initializes an ArrayList that is used to keep track of what indices have been checked in $\sigma_{\mathbb{O}}$. Then, the program sets the initial index to 0. Nested for-loops are used, the first one making sure the loop does not go through each element in σ_O more than once. The current index is added to the ArrayList of tested indices, and a new integer is initialized. This integer is the n^{th} (where n is the index) integer in $\sigma_{\mathbb{X}}$, and is the next number we need to look for in $\sigma_{\mathbb{O}}$. The next for-loop checks the elements in $\sigma_{\mathbb{O}}$ until it finds the integer we are looking for. Another for-loop checks the ArrayList of tested indices to check if the current index of $\sigma_{\mathbb{O}}$ has already been counted. If it has, the function terminates and tells the user that the grid diagram is that of a link. If not, the new index is set to that of the index of the integer we were looking for in $\sigma_{\mathbb{O}}$, and the outermost for-loop repeats. If the for-loop completes without finding a link, the program terminates and tells the user that the grid diagram is a knot.

```
import java.util.ArrayList;

public class TestForKnot {
public static String testForKnot(int[] sigmaO, int[] sigmaX){

//Check if arrays are of the same length
if (sigmaO.length != sigmaX.length){
return "Please input arrays of the same length";
}

//initialize array-list to keep track of where we have looked before.
ArrayList<Integer> tested=new ArrayList<Integer>();

//begin at index 0
int index=0;


//Don't go out of the bounds of SigmaO
for (int n=0; n<sigmaO.length-1; n++){

//add current index to tested
tested.add(index);

//Look for the sigmaX[index] in sigmaO
int toLookFor=sigmaX[index];

//Check for toLookFor in sigmaO. Once you find it, set the index to that so the cycle
//can continue.
for (int count=0; count<sigmaO.length; count++){
if (sigmaO[count]==toLookFor){
for (int element : tested){
if (element==count){
return "Link";
}
else{
index=count;

}
}
}
}
}
return "Knot";

}
```

3.1.2. *Calculating Tau.* As the tau invariant is what we have been researching this summer, we created a program that uses the shortcut to calculate Tau given the X permutation of a given diagonal grid diagram. We only needed the X permutation as an input, as the shortcut for tau will only work on diagonal grid diagrams so we know that the O's will be on the diagonal and we know the size of the grid diagram from the size of the X permutation. This allows us to construct the entire grid diagram using arrays of strings once we knot the X permutation.

In order to calculate tau we created two functions, topRight and bottomLeft, which both take in four arguments: the array of strings that is the grid diagram, the row and column that we are starting the search from, and the particular string we are looking for. The functions then count the number of particular strings, such as X or O to the top right and bottom left of the starting position using for loops to search through the grid and return the number of such strings.

Once we had these functions, we were able to calculate $\mathcal{J}(\mathbb{X},\mathbb{X})$, $\mathcal{J}(\mathbb{X},\mathbb{O})$, $\mathcal{J}(\mathbb{O},\mathbb{X})$, and $\mathcal{J}(\mathbb{O},\mathbb{O})$ relatively easily, as all we needed to do was use for loops to find the X's or O's, then call topRight and bottomLeft on each instance of X and O and add the outputs. Once we created code for each of these, we realized that the code for $\mathcal{J}(\mathbb{X},\mathbb{O})$, $\mathcal{J}(\mathbb{O},\mathbb{X})$, and $\mathcal{J}(\mathbb{O},\mathbb{O})$ was redundant, as $\mathcal{J}(\mathbb{X},\mathbb{O})$ will always equal $\mathcal{J}(\mathbb{O},\mathbb{X})$ and $\mathcal{J}(\mathbb{O},\mathbb{O})$ will always be zero, thus eliminating them all from the shortcut tau equation.

The more difficult part of calculating tau was calculating $\mathcal{J}(\mathbf{x},\mathbb{X})$ and $\mathcal{J}(\mathbf{x},\mathbb{O})$, as the set of points in $\mathbf{x}$ are not in the cells of the grid but rather on the intersection of grid lines. In order to calculate $\mathcal{J}$ for these points, we ran the function topRight from the cell to the bottom left of the point in $\mathbf{x}$ and the function bottomLeft from the cell to the top right of the point. This works because topRight and bottomLeft do not count the row and column that you start in, just the cells to the top right and bottom left, respectively. This works for all the points in $\mathbf{x}$ except for the one in the bottom left corner, but this point will always have every X and O to the top right of it.

Once we used topRight and bottomLeft to get all the counts we needed for the shortcut tau equation, we divided each count by two to get $\mathcal{J}$ and substituted the appropriate values into the equation to calculate tau for a given grid diagram.

3.1.3. *Simplifying Grid Diagrams.* In order to make classifying diagonal knots easier, we decided it would be best to use a program to simplify grid diagrams. We started with a program created by Park Mikels and Orlando Guerra [**?**] that used commutations in order to find adjacent X's and O's in the grid diagram, leading to a destabilization opportunity. Their program, however, did not include cyclic permutations when searching for adjacent X's and O's. We added code that performed cyclic permutations and looped through their code again with the cyclically permuted grid diagram, which improved the number of grid diagrams the program could simplify slightly.

3.1.4. *Generating Diagonal Grid Diagrams.* In order to see how diagonal grid diagrams, the only grid diagrams that the shortcut for tau can be used on, we decided to create a program that would generate all of the X permutations of diagonal grid diagrams of a given size, n. Once we had generated these we could calculate tau and other invariants of these knots to compare the knots and determine what kind of knots can be represented on a diagonal grid diagram.

Our first attempts to generate all the diagonal grid diagrams of size n were rather inefficient. As n got larger, our programs would take much longer to run and the output would be too large to handle effectively.

The code for our first attempt generated the diagonal grid diagrams randomly by shuffling the X permutation (5 4 3 2 1), checking if this permutation makes a knot using TestForKnot, then adding it to a list of X permutations if we have not generated it already. We realized that this was inefficient if we wanted to generate all diagonal grid diagrams of size n, but we later adapted it to generate a given number of random diagonal grid diagrams for large n, such as $n = 100$, which our other programs that generate all diagonal grid diagrams cannot handle.

Our next program generated every permutation of $\sigma_{\mathbb{X}} = (1\ 2\ 3\ \ldots\ n)$ recursively, including those that do not create knots on a diagonal grid diagram, then used TestForKnot to determine which of these permutations were knots and stored them in an array. This was also impractical, however, as we were generating $n!$ permutations when only $(n-1)!$ were knots that we could use the tau shortcut on.

The last program we created generated all X permutations that were knots systematically using recursion. First, the function diagonalKnots takes in the size of the grid diagram you want to generate, then creates a grid diagram using arrays of strings that has O's on the diagonal but no X's. It then removes the top row of this grid diagram and uses the ArrayList variables rowRemoved and colRemoved to keep track of which rows and columns have been removed and calls the helper function diagonalKnotsHelper on this grid diagram.

This helper function takes in a grid diagram that has a row removed, so that it is an $n-1$ by n grid. The helper function finds the column that does not contain an O, then enters a for loop. The for loop adds an X to this column, starting with the top row, then adds the X into the appropriate position in another grid that has not had any rows or columns removed using rowRemoved and colRemoved to keep track of which row and column the X would actually be in without rows and columns removed. The function then removes the row and column that contains this X, keeping track of which row and column it removed by adding them to rowRemoved and colRemoved and calls the helper function on this new grid. This process is illustrated in Fig. 4. After the helper function calls itself, it then resets the changes it just made to the grids, adding the row and column back in and removing the X from both grids.

Before the function enters the for loop, it checks to see if we have added enough X's to the grid that has not had any row or column changes, and if we have then we add this X permutation to a list of X permutations. This allows us to generate a list of every X permutation of diagonal grid diagrams that make knots without having to generate the additional unnecessary X permutations.

3.1.5. *Alexander Polynomial.* The last program we decided to create was one that calculated the Alexander polynomial of a given knot based on its grid diagram representation. We coded this in both Java and Mathematica, as Mathematica is able to calculate determinants of matrices much faster than our Java program. Both programs use the process outlined in **??** to calculate the Alexander polynomial.

They start by finding the winding number of the knot at every required point in the grid diagram by counting the number of vertical strands to the right of the point and noting the orientation of each. The orientation of each vertical line is found by noting whether the X

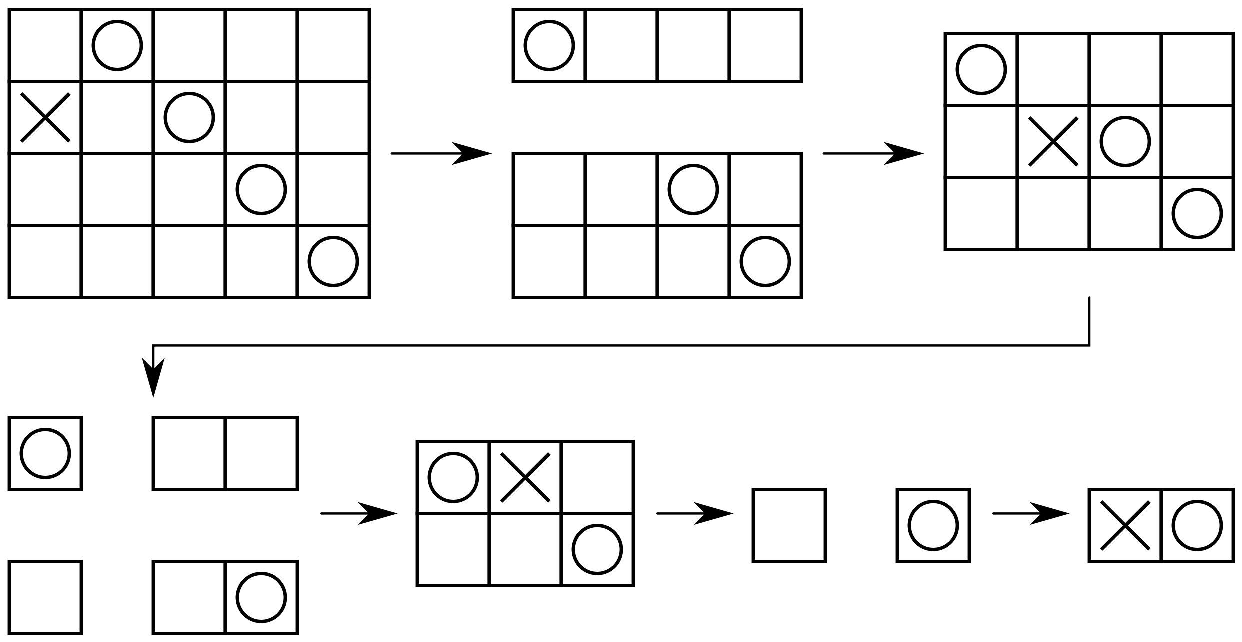


FIGURE 4. Recursive method for generating diagonal grid diagrams. Note: The position of the X's is chosen arbitrarily in this example.

is above or below the O in that column. The negative winding numbers are then used to create powers of t in the minesweeper matrix.

Once the program has found the minesweeper matrix, it simplifies the matrix and calculates the determinant, which is the Alexander polynomial multiplied by a power of t. The programs factor this power of t and return the Alexander polynomial of the knot.

3.2. **Non-Negativity of τ for Diagonal Knots.** While using Sarkar's shortcut to calculate τ, we notice that all of the computed values are greater than or equal to zero. We are able to spot check the τ values for various grid sizes and for particular knots; however it is not feasible to compute this invariant for every possible diagonal knot. We have proven that the τ invariant will always be non-negative.

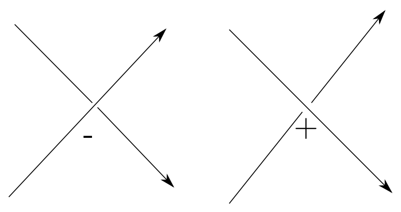


FIGURE 5. Examples of a positive and negative crossing.

Definition 2. A crossing in a knot is a *positive crossing* if the crossing follows the right hand rule. Examples of positive and negative crossings can be seen in Fig. 5.

Definition 3. A knot K is considered to be a *positive knot* if there exists a diagram for K that contains only positive crossings.

For example, all three of the crossings in the standard diagram of the right handed trefoil (Fig. 6) are positive, so the trefoil is a positive knot.

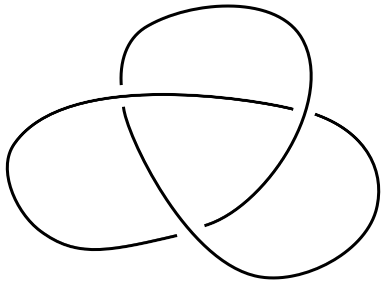


FIGURE 6. The standard diagram for the right handed trefoil contains all positive crossings, and therefore the right-handed trefoil is a positive knot.

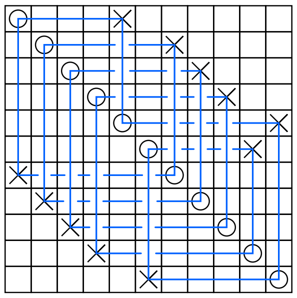


FIGURE 7. A diagonal knot that is not a torus knot.

Theorem 3. *Every knot that can be represented by a diagonal grid diagram is a positive knot.*

Proof. In a grad diagram, the X's connect vertically to the O's and the O's connect horizontally to the X's. Because the vertical strand is always the overcrossing and the O's are assumed to be on the diagonal, all of the crossings in a digaonal grid diagram will follow the right hand rule. □

4. The class of diagonal knots

As we examined the output of our programs, we saw that for small grid sizes, all of the diagonal grid diagrams we generated represented a torus knot or a connect sum of torus knots. This initially led us to guess that all diagonal knots are torus knots or connect sums of torus knots. However, as we generated larger grid diagrams, we did find one diagonal grid diagram that does not represent a torus knot. This grid diagram is shown in Fig. 7.

Theorem 4. *There exists a diagonal knot which is neither a torus knot nor a connect sum of torus knots.*

Proof. The first knot that is represented diagonally that is not a torus knot or a connect sum of torus knots that we have found is represented with grid size $n = 11$. The X permutation is $\sigma_{\mathbb{X}} = \{5, 4, 3, 2, 11, 1, 1, 10, 9, 8, 6, 7\}$. This knot has a τ invariant of 9 and an Alexander Polynomial of $t^{18} - t^{17} + t^{14} - t^{13} + t^{12} - t^{11} + t^{9} - t^{7} + t^{6} - t^{5} + t^{4} - t + 1$.

This polynomial does not match that of any torus knots that are represented diagonally with a grid size of 11. Therefore, we can confidently say, assuming that we have calculated the polynomial correctly, that the aforementioned knot is something other than a torus knot or a connect sum or torus knots. □

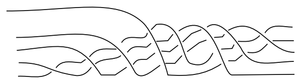


FIGURE 8. The braid form of the non-torus knot.

Initially, we thought that Fig. 7 was a twisted torus knot, as it looks similar to a torus knot, but does not fit the definition of the topmost strand wrapping over all the lower strands.

However, the knot does not satisfy any of the definitions for a torus links or twisted torus knots we were able to find. Therefore, we will make a new definition for a class of knot that this knot satisfies, based on the definition of a t-link from [BK09].

Definition 4. A *general twisted torus link*, denoted $T((r_1, s_1), (r_2, s_2), ..., (r_k, s_k))$ is the link defined by the closure of the braid $\mathbb{T} = (\sigma_1\sigma_2...\sigma_{r_1-1})^{s_1}(\sigma_1\sigma_2...\sigma_{r_2-1})^{s_2}...(\sigma_1\sigma_2...\sigma_{r_k-1})^{s_k}$ where $0 < s_i$, $i = 1, ..., k$.

Our definition for a general twisted torus link is the same as Birman and Kofman's from [BK09], except we do not require $2 \leq r_1 \leq r_2 \leq ... \leq r_k$. Fig. 9 shows the knot in general twisted torus link form.

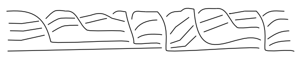


FIGURE 9. The knot in a form satisfying the definition for a general twisted knot.

In order to find exactly which set of knots can be represented diagonally, we turn to the braid representation of knots.

4.1. Braids and Knots.

Theorem 5 (Alexander's Theorem [Ada94]). *Every knot or link is the closure of a braid.*

Likewise, the closure of every braid is a knot or link. Using the logic used later in **??**, we realized that a knot diagram recovered from a diagonal grid diagram will always contain only positive crossings.

Since we know that the knot diagrams obtained from diagonal grid diagrams will always have positive crossings, we know that

$$\{\text{Diagonal Knots}\} \subseteq \{\text{Positive Knots}\}.$$

However, we know that the set of positive knots is not a subset of diagonal knots. A counterexample is the figure-eight knot, whose grid diagram cannot be made diagonal.

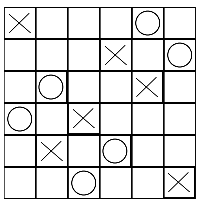


FIGURE 10. The grid diagram of a figure-eight knot.

5. Conjectures and Future Work

After comparing the list of knots represented on the diagonal grid diagrams generated by our program to the information found in the online Knot Atlas*****add citation*****, we realized that all of the knots that our program generates are positive braid knots.

Conjecture 1. A knot can be represented diagonally if and only if it has a positive braid representation.

The statement in Conjecture 1 is equivalent to

$$\{\text{Diagonal Knots}\} = \{\text{Positive Braid Knots}\}.$$

Definition 5. A *positive braid* is a braid consisting only of positive crossings.

Definition 6. A *positive braid knot* is the closure of a positive braid.

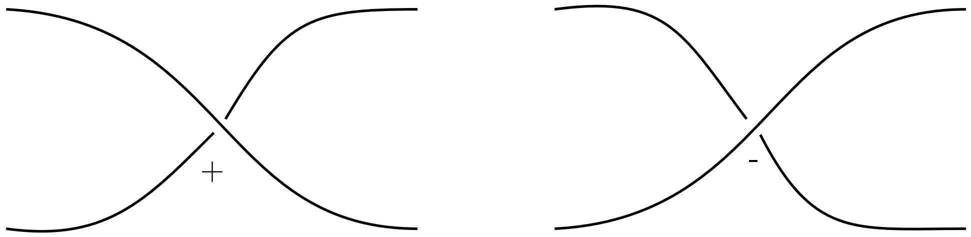


Figure 11. Examples of a positive and negative braid crossing respectively.

When investigating which types of knots can be represented diagonally, we initially hypothesized that all positive braid knots can be represented diagonally. We knew torus knots could be represented and we found an example of a twisted torus knot, both of which are subsets of positive braid knots. In order to prove this, we attempted to prove that

$$\{\text{Diagonal Knots}\} \subseteq \{\text{Positive Braid Knots}\}.$$

We have shown that most diagonal grid diagrams can be manipulated into positive braids. We found a method, stated in [NT08], to recover a braid diagram from a grid diagram by connecting any X that is to the left of an O to the left off of the grid diagram, as shown in Fig. 12. However, this can create negative crossings below the diagonal. Therefore, the braid will not always be positive and we have yet to find a algorithm that always makes the braid positive.

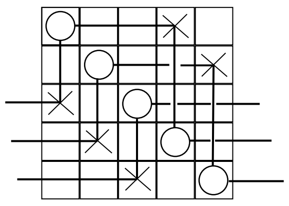


Figure 12. Connecting the X's and O's so that a braid is formed from the grid diagram of the trefoil.

The negative crossing is created whenever an X is to the top right of another X when they are both still below the diagonal. We attempted to eliminate the negative crossings by performing commutations on the row that creates the negative crossing. This is possible because the X and O will always be nested in between the X's and O's in some of the rows below it because of the position of the X's as well as the O's on the diagonal.

This method can create negative crossings in other parts of the grid diagram, however, as the O's will no longer be on the diagonal after the commutations are performed. We have yet to find an algorithm that eliminates the negative crossings in the grid diagram consistently.

We have also noticed a pattern in the braids formed using this method. The braid will always consist of a series of negative crossings followed by a series of positive crossings. We can divide a braid word into groups, with each group consisting of strictly decreasing consecutive braid letters, for example

$$\sigma_4\sigma_3\sigma_2\sigma_1\sigma_3\sigma_2\sigma_1\sigma_4\sigma_2\sigma_3\sigma_2 = (\sigma_4\sigma_3\sigma_2\sigma_1)(\sigma_3\sigma_2\sigma_1)(\sigma_2)(\sigma_3\sigma_2).$$

If we group the positive section of the braid in this way, we notice a pattern in the last braid letter of each of these groups.

If we create a sequence containing the last braid letter from each of these groups, then that sequence will always be decreasing.

We suspect that there is a similar pattern in the section of the braid diagram that contains negative crossings, as this can be thought of as a rotation of the grid diagram.

In yet another failed attempt to prove that diagonal knots are a subset of positive braid knots, we perform braid isotopies on braids of this pattern in order to eliminate the negative crossings, creating a positive braid. Again, we could not find an algorithm that consistently eliminates the negative crossings using braid isotopies.

Another method we used to recover a positive braid from a diagonal grid diagram involves splitting the strands at specific points. We connect the X's and O's in the grid diagram in the conventional way. If we divide the strands at every O that contains an X directly below it and an X directly to the right, then we can usually recover the braid diagram. Start in the top left of the grid diagram, then follow the strands as they move around the grid diagram clockwise, noting each strand that passes over multiple strands as its own decreasing consecutive group of braid letters. This method does not work, however, if there are any O's that contain an X above it and to the left in a position that is left of an O with an X below and to the right of it.

We initially thought that diagonal knots are a subset of positive braid knots while attempting to prove that the set of all diagonal knots is equivalent to the set of all positive braid knots, but we have yet to show that all positive braid knots can be represented diagonally. We have, however, found some restrictions on positive braid knots that ensure that the knot can be represented diagonally.

First, we divide the braid word into groups, with each group consisting of strictly decreasing consecutive braid letters.

We then list the last letter of each of these groups in a new sequence of braid letters. If this new sequence of letters has four or fewer local extrema, then the closure of the positive braid formed by the braid word can always be represented diagonally using our method.

Our method consists of splitting the braid diagram at the maxima, then putting one of the resulting parts of the grid diagram in the top right of a diagonal grid diagram and the other part in the bottom left. In order to put the braid in the grid diagram, start with parallel strands, then whenever a crossing occurs drop the strand vertically over the other strands, making a square braid diagram. Then place an O at the start of each strand and place O's and X's at every point a strand goes from vertical to horizontal or vice versa, ending each strand with an O at the end of a vertical strand. The resulting grid of X's and O's should have the O's on the diagonal. This same process can be repeated for both sections of braid, then attached by rotating one so that it fits into the bottom left of the grid diagram.

It is also important to note that the connect sum of any positive braid knot that is represented this way can also be represented on a diagonal grid diagram, as the connect sum of two knots that can be represented on a diagonal grid diagram can also be represented on a diagonal grid diagram.

6. Acknowledgements

This research was funded by Dr. Albert H. and Greta A. Bryan through the 2017 Bryan Summer Research Program at Simpson College. We are grateful to Park Mikels and Orlando Guerra, who wrote a program as part of their Spring 2017 senior capstone project that was very helpful as we began writing our grid diagram programs.

References

[Ada94] Colin C. Adams, *The Knot book*, W.H. Freeman and Company, New York, 1994.

[BK09] Joan Birman and Ilya Kofman, *A new twist on Lorenz links*, arXiv:0707.4331v4 (2009).

[MOST07] Ciprian Manolescu, Peter Ozsváth, Zoltán Szabó, and Dylan Thurston, *On combinatorial link Floer homology*, Geom. Topol. **11** (2007), 2339–2412. MR 2372850 (2009c:57053)

[NT08] Lenhard Ng and Dylan Thurston, *Grid diagrams, braids, and contact geometry*, arXiv preprint arXiv:0812.3665 (2008).

[Sar11] Sucharit Sarkar, *Grid diagrams and the Ozsváth-Szabó tau-invariant*, Mathematical Research Letters **18** (2011), no. 6, 1239–1257.