A NEW METHOD OF KNOT COUNTING

Kelsie L. McCartney

This thesis is submitted in partial fulfillment of the requirements of the Research Honors Program in the Department of Mathematics and Computer Science

Marietta College

Marietta, Ohio

April 20, 2008
This Research Honors thesis has been approved for
the Department of Mathematics and Computer Science and
the Honors and Investigative Studies Committee by

______________________________  _____________
Dr. John Tynan: Faculty thesis advisor
Date

______________________________  _____________
Dr. Mark Miller: Thesis committee member
Date

______________________________  _____________
Dr. Matthew Menzel: Thesis committee member
(if applicable)
Date
Acknowledgements

I would like to thank Dr. John Tynan, my thesis advisor, for all of his assistance and guidance during the process of my research. Without his invitation to take on this project, his willingness to work with me, and his encouragement throughout, this endeavor would not have been successful. I would also like to offer my appreciation to Dr. Mark Miller and Dr. Matthew Menzel who made up the remainder of my thesis committee. Their assistance and feedback have played a large role in my success with knot counting. Additionally, I am grateful to the entirety of the mathematics department for their continuous acceptance and support throughout my time at Marietta College. Lastly, I would like to express my gratitude to my friends and family who always stand by me in all that I do.
# Table of Contents

Abstract 1
Relevance and Overview 2
Knots 101 2
Matrix Representation 4
Rules 6
  - Basic Knot Construction 6
  - Column/Row Equivalence 7
  - Singular Values 10
  - Type II Reidemeister Move 13
  - Type III Reidemeister Move 14
  - Links 17
  - “Trapped” Strands 19
Another Approach (The Over/Under Method) 20
Collaboration 23
  - Matrix to Over/Under 24
Rule Overview 24
  - Matrix Rules 25
  - Over/Under Rules 25
The Algorithm 26
Conclusion 29
References 31
Abstract

In knot theory, the exact number of unique knots with seventeen crossings or more is unknown. This project used a matrix representation of knots to formulate an algorithm to count the number of unique knots of a given size. To eliminate over-counting, rules were written into the algorithm based on knot and matrix construction. Because only unique knots were desired, any matrix construction that presented an opportunity to perform a Reidemeister move had to be eliminated from the process. Similarly, matrices that did not create knots and those that created links needed to be excluded. A problem arouse with the matrices that did not produce knots called the “trapped” strand problem. This dilemma inspired the formulation of a new method of knot representation, the over/under method. While insufficient on its own, the over/under method provided an excellent supplement to the matrix representation. The two methods were used in collaboration to develop an algorithm to count knots. When appropriate decisions were made in matrix construction, the pre-rule over count of possible matrices for an $n \times n$ knot was lowered to $(n - 2)^n$. It has been proven so far that the algorithm works without error for knots with 3, 4, and 5 crossings. If this project is continued, the algorithm can be put into motion on a computer and its accuracy can be evaluated. From there, the process can continue, modifying the algorithm and improving the count.
Relevance and Overview:

In knot theory, the exact number of unique knots with three to sixteen crossings is known (there is 1 unique knot with three crossings and there are 1,388,705 unique knots with sixteen crossings). For knots with seventeen or more crossings, however, it is unknown exactly how many unique knots exist. This is an open question in knot theory. But the study of knots has additional relevance outside of mathematics.

Recently, knot theory has become increasingly interesting to biological fields, as knots provide a model for DNA. Knot theory may also be important to the development of quantum computers.

This project works toward the development of an algorithm to determine preliminary estimates for the number of unique knots of a given number of crossings using matrix representation.

Knots 101:

A mathematical knot can be thought of as a mass-less piece of string tangled up in itself with its ends glued together. In this way, the knot is closed; there is no way the string can pass through itself and change into any other knot. This is how uniqueness is determined. If the real-life string can be manipulated and deformed from one picture to another, then it is in fact the same knot.

These three knots are all the same, simply stretched or deformed from one to another. It is in this feature that a problem arises in knot theory: how does one tell if two knots are the
same? While there is no answer for this in all cases, there are some steps to consider. When considered thoughtfully, it can be seen rather easily that the first two knots in figure one are, in fact, the same knot. These two pictures involved the same strands interacting at the same crossings; the strands are just stretched and twisted up a bit more in the middle knot. When the picture of a knot is the deformation of another, as in the first two knots in the figure, they are called planar isotopes.\textsuperscript{1} The differences in the pictures of planar isotopes are generally visible and have no method of same-ness determination. The third knot, on the other hand, does change the strands and crossings. In this final knot, two additional strands and two additional crossings appear. There are ways to deal with these changes. The three ways that change the projection of a knot by altering the relationships between strands and crossings are called Reidemeister moves.\textsuperscript{1}

The first of the three Reidemeister moves, TYPE I, is a twist. When one twists (or untwists) a piece of a strand, it adds (or eliminates) a crossing, but does not change the knot:

![Figure 2: TYPE I Reidemeister move](image)

The second Reidemeister move, TYPE II, pulls one piece of a strand over another. The third knot in figure one is an example of this. This overlap creates more crossings than are necessary:

![Figure 3: TYPE II Reidemeister move](image)
The third and final Reidemeister move, TYPE III, involves pulling a strand over or under a crossing. This move does not change the number of crossings, but changes the relation between them.

Figure 4: Type III Reidemeister move

While all of these moves change the projection of a knot, none actually changes the knot. In this way, no matter how different two knots look; if they are the same knot, it is possible to get from one projection to the other by performing some series of Reidemeister moves. However, while it is possible, this method is generally not ideal. These moves pile up quickly, especially when the exact pattern is unknown (and if it were known, it probably would not need to be done). This is another motivation for this project; in addition to counting knots, another objective is to be able to look at two knots and determine whether or not they are distinct.

Matrix Representation:

In William Sears’ 2007 mathematics capstone paper, he investigated the $p$-colorability of knots. The technique of $p$-coloring is used to determine when two knots are the same. Sears used this technique to develop a new representation of knots. In his project, every distinct knot with three to nine crossings was broken down into the representation described below.\(^2\)

Using the concept of $p$-colorability, any knot can be written as a matrix. In order to do so, a knot is broken down into its crossings and the strands between the crossings (there will always be the same number of strands and crossings). For convenience, label the crossings with numbers and the strands with letters. To form the matrix that will define the knot, list the strands
across the top of the matrix and the crossings down the side. To fill in the rows, the crossings are considered; each crossing will have exactly one over strand and two strands that end there. The entry at the intersection of the over strand and the crossing is a negative two; the entry at the intersection of the under strands and the crossing is one.² In this way, the columns represent the strands and the rows represent the crossings. A few examples of this technique are demonstrated in the following figure.

![Matrix Representation of Knots](image)

Figure 5: The matrix representation of three simple knots
It is important to realize that with this technique, not only can a knot be turned into a matrix, but a matrix can also be turned back into a knot. From a given matrix, the columns are labeled with letters and the rows with numbers. The knot can then be drawn crossing-by-crossing placing the appropriate strand over the appropriate ending points.

So now, with these tools, it would seem that every knot can be written as a matrix and every matrix can be drawn into a knot. Unfortunately, this is not always the case. It is true that every knot can be written as a matrix, but not necessarily a unique one; multiple matrices will produce the same knot. Additionally, not every matrix will be able to produce a knot at all; some might have entries that do not make any sense at all while others might lead to situations where the strands in the final crossing are not in position to make the final connection. These are the situations that must be anticipated and corrected in order to get an accurate count of distinct knot matrices of a given size. By creating rules based on both the nature of knots and the properties of matrices, the next section of this paper attempts to do just that.

**Rules:**

*Basic Knot Construction:*

To begin with, there exist a few obvious rules the knot matrices must follow. First consider the allowed entries: each intersection must necessarily consist of an over strand and two under strands.

![Figure 6: A crossing: the intersection of two strands in a knot](image)

It is possible, however, to conceive of a crossing where the same strand is both the over strand and one of the under strands.
This construction is obviously eliminated by a TYPE I Reidemeister move. Note that this does not mean that this construction cannot happen or cannot be represented in matrix form; it can. However, the goal is to count the number of distinct knots, therefore a knot including this loop will have already been accounted for. To avoid double counting, this possibility will be eliminated from the list of created matrices.

With the elimination of all possible TYPE I moves, only the first image is valid. Thus the only possible entries for a knot matrix are -2 for the over strand in a crossing, 1 for the two ends, and 0 for a strand without involvement in a particular crossing. From this realization, not only are possible entries revealed, but stronger stipulation for row content arises. Each row in the matrix represents an individual crossing in the knot. It was just shown that each crossing has exactly one over strand and exactly two ends meeting there to form the under. Therefore, each row will have exactly one negative two, exactly two ones, and exactly $n - 3$ zeros in an $n \times n$ matrix.

Similarly, a rule can be obtained for the columns by strand inspection. Each column represents an individual strand in the knot. While it is not required that a strand cross over a certain number of other strands, it is true that a strand has exactly two ends (this is what makes it a strand). Hence, because every strand has exactly two ends, every column in the matrix must have exactly two ones.

**Column/Row Equivalence:**

Beyond the basics, the first thing to realize with this approach is that the power of choice creates duplication. When transforming a pictorial knot into its matrix representation, the matrix
maker has the choice to label the strands as well as the crossings however he might chose. Knots A and B in figure 8, for example, are obviously the same knot; it is the same exact picture. But by simply changing the labels on the strands and crossings from one to the other produces matrices that do not look alike. Is there some feature these two matrices share that signify that they are representations of the same knot?

As it turns out, there is a way. The labeling duplications from above occur only when the names of strands and crossings are switched and interchanged. But switching the names of the strands is just a switch of the columns that represent them. Similarly, switching crossing labels is just an interchange of rows in the matrix. So ultimately, these labeling duplications are just manifestations of some combination of row and column switches. To demonstrate, the moves to get from knot A to knot B and depicted: Start with matrix A from figure 8. First switch rows 1 and 2 (it should be noted that the steps can performed in any order).
Figure 9: The changes in knot A after switching rows 1 and 2

This switches the labels on those crossings. To get the rest of the crossings in the same orientation they are in knot B, move the second row to the fourth position, the third to the second, and the fourth to the third row (note that these steps can be accomplished by switching pairs one at a time until the effect is achieved).

Figure 10: The changes in knot A after the remaining row switches

Now all that remains is to interchange the strand names. Switching the first and last columns will switch a and d and interchanging the second and third columns will switch b and c creating knot B and thus completing the transformation.
Figure 11: After switching columns $a$ and $d$, matrix $A$ becomes matrix $B$

While it may seem clear that the matrices of any identical pictures with different labeling choices can be manipulated in this way to get from one to the other, it is still not obvious how one could look at two matrices alone and know that they create the same knot. One tool that can be utilized is singular values.

**Singular Values:**

The row and column switches involved in transforming one matrix to another of identical knots that differ only in labeling can be described simply by matrix multiplication operations. Left and right multiplying a matrix by specific elementary matrices switches the rows and columns in the matrix. From here on, the term elementary matrix will be used to denote the particular elementary matrices that create row and column switches. Elementary matrices are matrices that differ from the identity matrix only by row and column switches. To swap rows 2 and 3 of a $4 \times 4$ matrix, for example, multiply the matrix on the left by the elementary matrix $E_{2,3}$.

$$
E_{2,3} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
$$

**eq. 1**
Similarly, to swap columns 2 and 3 of a 4 x 4 matrix, multiply the matrix on the right by the elementary matrix $E_{2,3}$. The transformation from matrix $A$ to matrix $B$ then is

$$E_{1,2}E_{2,3}E_{3,4}AE_{4,1}E_{2,3} = B$$  \hspace{1cm} \text{eq. 2}$$

It follows, then, that any two matrices which are column/row equivalent (that is, any two matrices who create the same knot and differ only by strand and crossing labeling) can be written in this way; one matrix is left multiplied by a group of elementary matrices and is right multiplied by another group of elementary matrices to get to the other. Let $E_a$ represent the group of elementary matrices on the left and $E_b$ represent the group on the right. Then for any given column/row equivalent matrices $M$ and $N$, there exists $E_a$ and $E_b$ such that:

$$M = E_aNE_b$$  \hspace{1cm} \text{eq. 3}$$

It is important to note some features about elementary matrices. Since elementary matrices simply switch the rows or columns of the matrix they are multiplied by, if an elementary matrix is multiplied on itself, the switched rows get switched back and the identity matrix is produced. The result: every row-switch elementary matrix is its own inverse. This is true of all row-switch elementary matrices as well as any sequence of elementary matrices of this form. Additionally, because of its nature, interchanging any two rows of the identity matrix produces an identical result as interchanging the same two columns. In this way, it is evident that row-switch elementary matrices (as well as products of them) are their own transpose. To sum it all up:
\[ E = E^{-1} = E^T \]  

eq. 5

These properties of row-switch elementary matrices are extraordinarily useful in proving column/row equivalence. The subsequent argument follows from equation 3:

\[ M = E_a^{} N E_b^{} \]  

eq. 3

\[ M^T M = (E_a^{} N E_b^{})^T (E_a^{} N E_b^{}) \]  

eq. 6

\[ M^T M = E_{b}^T N^T E_{a}^T E_{a}^{} N E_{b}^{} \]  

eq. 7

\[ M^T M = E_{b}^T N^T E_{a}^{-1} E_{a}^{} N E_{b}^{} \]  

eq. 8

\[ M^T M = E_{b}^T N^T N E_{b}^{} \]  

eq. 9

\[ M^T M = E_{b}^{-1} N^T N E_{b}^{} \]  

eq. 10

The final relation in equation 10 is the relationship that defines similar matrices. Similar matrices are any matrices \( T \) and \( S \) such that for some invertible matrix \( X \),

\[ T = X^{-1} S X \]  

eq. 11

Therefore, it is evident by equation 10 that \( M^T M \) and \( N^T N \) are similar matrices. Since similar matrices have the same eigenvalues, this implies that \( M^T M \) and \( N^T N \) have the same eigenvalues. This becomes important when the definition of singular values in considered. For a square matrix, A’s, the matrix’s singular values are the square root of the eigenvalues of \( A^\dagger A \) (where \( A^\dagger \) is the conjugate transpose). Since any matrix representation of a knot can only have real values, \( M^\dagger = M^T \) for all knot matrices. Therefore, since for two column/row equivalent matrices \( M \) and \( N \), \( M^T M \) and \( N^T N \) have the same eigenvalues, this necessarily implies that \( M \) and \( N \) have the same singular values. Alas, the result is realized: column/row equivalent matrices share the same singular values.
The argument, however, cannot be followed in reverse to prove that two knot matrices with identical singular values are necessarily the same knot. Unfortunately, this is not the case. Ultimately, singular values cannot be used to eliminate matrices that duplicate knots. They can, however, assist in finding those duplicates. When two matrices have the same singular values, column/row equivalence should be checked; only when singular values are equal is it possible that two matrices are column/row equivalent.

**TYPE II Reidemeister Move**

Labeling the same picture differently is not the only way the same knot can be counted multiple times. Recall that when Reidemeister moves are performed on a knot, the knot itself does not change but the relation between strands and crossings does. This means that the knot does not change, but the matrix representation does. The goal is to count the number of distinct knots of a given size in their simplest form; therefore, if a Reidemeister move can be performed to lessen the number of crossings in a knot, it should be done before the knot is counted. For this reason, it is ideal to eliminate the possibility of constructing matrices on which a Reidemeister move can be applied.

The first of the Reidemeister moves was already eliminated by the rules of basic knot construction, but consider a situation where a TYPE II Reidemeister move could be performed to lessen the number of crossings in a knot:

![Figure 12: Construction where a Type II Reidemeister move is possible](image)

Regardless of which strands and crossings were involved, the matrix would look something like the following:
Figure 13: The sub-matrix of the knot construction of figure 12.

This construction will always form this sub-matrix and vice versa. Therefore, it can be concluded that the sub-matrix in figure 13 is an undesirable construction and should be omitted from any possible knot matrices. The rule then, can be stated as follows:

\[ \text{If } a_{ij} = 1 \text{ and } a_{ir} = -2 \text{ where } j \neq r \]
\[ \text{and } a_{kj} = 1 \text{ then } a_{kr} \neq -2 \text{ for } i \neq k \]

*Type III Reidemeister Move*

The third type of Reidemeister move is by far the most complicated. With this move, the knot does not gain or lose crossings; instead, it always remains the same size. For this reason, instances in which TYPE III moves can be performed cannot be eliminated. However, much like singular values are used to identify column/row duplicates, it is possible to recognize duplication of at least one type.

When a matrix has two columns that each have two negative twos, two kinds of TYPE III moves can be performed and always change the matrix in a predictable way. Consider the following knot:
Figure 14: The projection and matrix of a knot on which a Type III move can be performed.

This knot has multiple opportunities for TYPE III moves. One of them is pulling strand $c$ over crossing 5. When care is taken in naming the post-move knot, this move only changes the entries of two rows.

Figure 15: The post-move projection and matrix when strand $c$ is pulled over crossing 5.
In this kind of TYPE III move, both the 1’s and the -2’s in the involved rows change positions. Another TYPE III move that can be made on the original knot is pulling strand \( g \) under crossing 6. Again, with careful choice, this move only changes two rows as well.

<table>
<thead>
<tr>
<th>( g )</th>
<th>( a )</th>
<th>( b )</th>
<th>( c )</th>
<th>( d )</th>
<th>( e )</th>
<th>( f )</th>
<th>( g )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 16: The post-move projection and matrix when strand \( g \) is pulled under crossing 6.

In this kind of move, only the -2’s changed position in the two rows involved. As it turns out, these changes are systematic and can always be predicted. Two ways a TYPE III move can change a matrix are described in the remainder of this section.

The first of these kinds is the “under TYPE III.” Figure 16 is an example of this move. To get from the original matrix to the post-move matrix, start by selecting a column with no -2’s. In the above example, column \( g \) was selected. The two rows that will undergo alterations are the two rows that have their 1’s in this column. The two 1’s in column \( g \) are in rows 3 and 5, so these are the two rows that will change. For the under method, the 1’s stay in the same positions, but the -2’s change. To determine how the -2’s relocate, examine the -2’s in the two selected rows. If one of the -2’s is in a column with another -2, it will stay in the same column, but switch rows. In the example, the -2 in row 3 (which is also in column \( c \)) was chosen. It was moved to position \( c5 \). Placement of the final -2 is the trickiest. The column that was just used because it had two -2’s is again examined. In that column, the -2 that is not in either of the
changing rows is located. In the example, this -2 is in position c6. It is the row of this -2 that will be used for the final placement. The last -2 will have started in a column in common with a 1 in this row. It must be moved to the column of the other 1.

This sounds confusing, but consider the example. The -2 in position c6 was being examined. In row 6, the 1’s are in columns b and e. In the original matrix, the -2 in row 5 is already in column e, so it must move to column b. And of course, since every row must have exactly one -2, it will be in row 3.

A similar technique is used for the “over TYPE III” move. Figure 15 shows an example of this change. Again, start by selecting a column with no -2’s (g in the example). As before, the two rows with the 1’s in this column are the only rows that will change (rows 3 and 5). This time the 1’s switch rows. Select a -2 in one of these two rows the same way as before, select one that shares its column with another -2 (the -2 in position c3). For this process, that particular -2 remains in its original location. To place the second -2, consider the row of the other -2 in the same column (row 6). Of the 1’s in this row, one of them will be in a column with no -2’s (column b). This is where the other negative will be placed (b5).

The processes seem similar, and in fact, one might notice that the two techniques produced the same two new rows, but just exchanged their placement. This is column/row equivalence! While the original matrix is not column/row equivalent with either kind of TYPE III Reidemeister move, the knots that result after pulling strand c over crossing 5 and strand g under crossing 6 are column/row equivalent. This means that only one of the processes described above is required to eliminate some TYPE III Reidemeister move duplications.

*Links:*

The creation of matrices that duplicate knots is not the only phenomenon to worry about; there are other constructions that must also be eliminated. One such construction is that which produces an assortment of links instead of making a knot. Consider the following matrix:
Figure 16: A matrix of a link

While the matrix does not break any rules and can be created in pictorial form, once the picture is drawn, it is evident that it does not create a knot, but rather a link. A knot requires that it can be constructed with a single, unending piece of string. A link, on the other hand, is created from multiple pieces of string. A link is multiple knots tangled together. The two separate pieces in the creation of the matrix in figure 16 can be seen in the following figure:

How can a link matrix be identified? Because a link consists of more than one piece of string, the strands that make up one string in the link as should somehow be grouped together. If any subset of columns in a matrix contains either two 1’s or zero 1’s in each of its rows, then that matrix forms a link. This follows naturally from the fact that when a link is traced, it will come back to itself and possibly never reach some of the crossings in the form. From the matrix above, because the columns can be grouped as described, it is evident that strands $a$, $d$, and $e$
form one string in the link while $b$ and $c$ form the other.

\[
\begin{array}{cccc|cccc}
 & a & d & e & b & c \\
\hline
\text{No Ones} & 1 & -2 & 0 & 0 & & & \\
\text{Both Ones} & 2 & 1 & 1 & 0 & & & \\
\text{Both Ones} & 3 & 1 & -2 & 1 & & & \\
\text{No Ones} & 4 & 0 & 0 & -2 & & & \\
\text{Both Ones} & 5 & 0 & 1 & 1 & & & \\
\end{array}
\]

Figure 18: The sub-matrices that make up the two individual knots in the link

Whenever the situation in a matrix arises when some subset of columns contains both or neither 1 in each of its rows, then the matrix can be eliminated.

“Trapped” Strands:

Another instance of matrix construction that will ideally be avoided is that of a knot that cannot be created. For lack of a better term, a strand is called “trapped” when it cannot reach the other strands with which it needs to meet up to properly complete a crossing. For example:

\[
\begin{array}{cccc|cccc}
 & a & b & c & d & e \\
\hline
\text{No Ones} & 1 & 1 & -2 & 1 & 0 & 0 & \\
\text{Both Ones} & 2 & 1 & 0 & -2 & 0 & 1 & \\
\text{Both Ones} & 3 & -2 & 1 & 1 & 0 & 0 & \\
\text{No Ones} & 4 & 0 & 0 & -2 & 1 & 1 & \\
\text{Both Ones} & 5 & 0 & 1 & 0 & 1 & -2 & \\
\end{array}
\]

Figure 18: A matrix that creates a “trapped” strand problem
In this knot, when the crossings are drawn one by one, the final crossing cannot be created. In order to complete the knot, strands $b$ and $d$ need to meet under strand $e$. But that is simply not possible. It does not matter in which order the crossings are drawn; the result is always the same: the knot cannot be completed. Some knots even run into trouble before the final crossing is attempted.

Because the order of creation does not matter, sometimes the problem is that strand $b$ cannot meet up with strand $d$, and other times, $a$ and $c$ or some other strand pair creates a “trapped” problem. For this reason, this problem is difficult to pinpoint. At this point, there is still no steadfast rule for identifying and eliminating matrices that create a “trapped” strand and therefore do not create a knot. From this dilemma, another strategy was sought.

**Another Approach (The Over/Under Method):**

The “trapped” strand issue, along with other challenges in minimizing the distinct knot count led to another method of representing knots. This method is not identical to, but is inspired by Dowker notation.\(^1\) In the over/under method, a crossing in the knot is selected at random and labeled one. From there on, the string is followed and each crossing is labeled with the next number. When the strand being followed goes over a crossing, that number becomes the over number. Similarly, when the strand being followed goes under a crossing, that number becomes the under number. This process continues until the trace comes back to the beginning and all crossings have both and over and an under number.
The over/under pairs are an alternative representation of the knot. It is simple to get from the over/under pairs back to the knot by simply putting the pieces together in numerical order.

This representation has its own set of rules. The first rule, naturally is that each pair must include one over and one under; a result from the nature of a crossing. Additionally, if consecutive numbers are paired in the same crossing, the only possible explanation is that a TYPE I move can be conducted to eliminate it.

Therefore, these combinations can be eliminated. Other pair possibilities can also be considered. If the numbers in a pair differ by only 2, a trapped strand would be created. Regardless of whether the strand is an over or an under, it is impossible for the strand to escape from inside the loop (if it did, it would add a crossing between the over and under, changing the difference).
Figure 21: When pair difference is 2, it results in a “trapped” strand

From this, it can be concluded that pairings that differ by 2 are not allowed. However, it is not just pairs that differ by 2. It quickly follows that pairs that differ by any even integer are not permissible.

Figure 22: When pair difference is any even number, it results in a “trapped” strand

It is possible that an even number of the pieces connect, but there will always remain one strand that is trapped. Hence, it is impossible to create a knot out of any over/under representation that includes a pair with similar parity. All pairs must include one even number and one odd number.

Along similar lines, the final rule so far involves pairings that differ by 3. For a pair with difference 3, there are exactly two crossings on the loop:

Figure 23: The construction when the pair difference is 3
The only possible way this structure can form a legitimate knot is if the two strands involved in the crossings in the loop connect with each other.

![Figure 24: When pair difference is 3, the strands must connect](image)

This means, when a crossing is made up of a pair that differs by 3, the integers between the numbers in the difference 3 pair must be paired with consecutive integers. For example, in figure 24, if the pair at the blue crossing is 3 and 6, then 4 and 5 must be paired with consecutive integers. So 11/4, 5/10 would be acceptable with 3/6; the pairs 11/4, 5/16 would not be acceptable and that knot could be thrown out.

In this representation, just like in matrix representation, duplication due to labeling decisions can occur. Both the choice of where to begin labeling and in which direction to travel can be made in this method, creating multiple representations of the same knot. These duplications can be handled simply and systematically by adjusting each of the numbers in a representation in the same way and comparing to other representations.

**Collaboration:**

On its own, the over/under method is tedious and impractical. In order to create the over/under pairs without a picture of a knot, the order of over strands and under strands must be known. This method certainly could not be used independently to get any sort of a reasonable count of distinct knots. However, the over/under method does tackle some problems the matrix approach could not deal with. If the two methods could somehow be combined, the result could produce
better results than either process on its own.

It is possible to translate between the matrix and over/under representations of a knot!

**Matrix → Over/Under**

To create over/under pairings from the matrix representation, start with the matrix. In place of the letters above the columns denoting the strands and the numbers to the left of the rows denoting the crossings, pairs of numbers are going to be issued instead. The specific entries will be assigned according to the following process:

First of all, in the top entry above the first column place a 2 (without loss of generality). To determine the number below it, add one plus the number of negative twos present in that column. This bottom number is also the top number of the next column. Proceed in this fashion, adding one plus the number of negative twos to the top number to obtain the bottom number until all columns are labeled (note: the bottom entry of the $n^{th}$ column should be two). Determination of the side numbers is a bit trickier. For assignment of the bottom number in each row, determine the common number of the two columns with ones in this row. This is the row bottom number. For the top number down the side, consider the column numbers; each skipped number becomes a row top number. The placement of the skipped numbers is decided by the location of the negative twos in the matrix. When the difference of the column number pair is 2, the skipped number goes in the row with the -2. When the difference is 3, the two skipped numbers go with the two rows with -2’s. Which goes with which is based on the parity rule; evens must be paired with odds and vice versa. If it is not possible to follow this rule, the knot cannot be created. Continue this process until all of the row pairs have been compiled. These row pairs are the over/under pairs in the over/under representation.

**Rule Overview:**

Since the ultimate goal of this project is to create an algorithm to count knots, the major rules of both representations is outlined below to provide a basis for the program.
Matrix Rules:

1. Basics
   - All entries $\in \{0, 1, -2\}$
   - Each row has exactly one $-2$, two $1$s, and $(n - 3)$ $0$s
   - Each column has exactly two $1$s

2. Same singular values $\Rightarrow$ column/row equivalence $\Rightarrow$ same knot

3. TYPE II
   - If $a_{ij} = 1$ and $a_{ir} = -2$ where $j \neq r$
   - and $a_{kj} = 1$ then $a_{kr} \neq -2$ for $i \neq k$

4. TYPE III
   - If it is possible to chose a column, $x$, such that $c_r \neq x \ \forall r$.

   If $c_x = c_y$ for some $y$, then $c_{x+1}' = c_x$ and if $y \neq x + 1$ then $c_x' = y$, else $c_x' = y + 1$

   If $c_{x+1} = c_y$ for some $y$, then $c_x' = c_{x+1}$ and if $y \neq x$ then $c_{x+1}' = y$, else $c_{x+1}' = y + 1$

   And the matrix with the only difference from the original being $c_{x+1}'$ and $c_x'$ in place of $c_{x+1}$ and $c_x$, then it is a duplicate

5. Links
   - If any subset of columns contain either 0 or 2 ones in each row $\Rightarrow$ it is not a knot

Over/Under Rules:

1. TYPE I
   - Pairs with difference = 1 cannot exist

2. Parity
   - All pairs must include one even and one odd number

3. Special Case
   - If pair difference = 3, the numbers between the difference must be paired with consecutive integers.
The Algorithm

At first, construction of an algorithm commenced in which the program stopped at each of the \( n^2 \) locations in the matrix and asked, what could this entry be? With each possible option, the program would then be instructed to create a branch for that option. This strategy would require massive amounts of computation and storage on a computer. Additionally, this method enables a pre-rule over count of \( (n^2)! \) knots.

Realizing the flaws in the plan, a more compact algorithm technique was sought. The next idea was to focus not on individual entries, but rather on rows. Instead of examining each and every entry in the matrix, consideration of row combinations could be used. Not only this, but since the choice is in the hands of the maker, the first row of the matrix could be defined: 1 - 2 1 0 0 0 … With this new method, the over-count and computation time are lessened, but the process can still be improved.

The best method to date goes beyond row combination consideration; the only choice now is -2 placement. With the power of labeling, the matrix maker has much more control over matrix decisions than just the first row. The matrix maker can actually place every single 1 into the matrix without making any assumptions about the knot. Arrange the 1’s as follows:

\[
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
& \ddots \\
1 & 1
\end{bmatrix}
\]

Figure 25: The matrix that will be used in the algorithm

The only thing this mandates about the knot is how one chooses to label it. Label the strands in order \( a, b, c \), etc. by tracing out the string. Label the crossings in the same fashion: crossing 1 is where \( a \) and \( b \) meet, crossing 2 is where \( b \) and \( c \) meet, …, crossing \( n \) is where stand \( a \) meets the
last strand. This can be done with any and every knot without loss of generality (the tactic actually helps eliminate some duplication). Now that the ones have been set, the only consideration is where to place the -2 in each of the rows. Automatically the pre-rule over count drops to \((n - 2)^n\). This count is exponentially smaller than the original estimate. Additionally, the computer no longer needs to store entire matrices; now all of the information about a single \(n\times n\) matrix can be found in only the \(n\) locations of the -2’s. Fortunately, the machine now has a more practical task. Pre-determined 1 placement eliminates the need for rules Matrix 1 and Matrix 5.

What is possibly more valuable is that the 1 placement allows all column/row equivalence to be eliminated. Now the only choice left to the matrix maker is where to start labeling a knot and then which direction to go from there. Both of these possibilities result in predictable changes in the locations of -2’s in the matrix and can therefore be anticipated and eliminated. With the abolition of every possible column/row equivalent matrix duplicate, singular values are no longer necessary as a tool of elimination.

The first step chooses the column of the negative two for each row. Notationally, let \(c_r\) give the column of the negative two in row \(r\). For example, if the negative two in row three is in the fifth column, then \(c_3 = 5\).

**Step 1:**

Pick \(c_1\) such that \(c_1 \neq 1,\) or 2
Pick \(c_2\) such that \(c_2 \neq 2, 3,\) or \(c_1\)

Pick \(c_m\) such that \(c_m \neq m, m + 1,\) or \(c_{(m-1)}\)

Pick \(c_n\) such that \(c_n \neq 1, n,\) or \(c_{(n-1)}\)

Repeat this step until every option is exhausted.

This step will create the matrices (not yet in matrix form, but by assigning the positions of the negative twos). After matrix creation, convert the matrices to their over/under representation.
Step 2: Transformation into and elimination by over/under representation

First we need a count, $t_c$, which represents the number of negative twos in a given column, $c$. This iteration can be added to the sequence of phases in step one.

When $c_n = c$, add one to $t_c$

Over/under pairs: $x_n/y_n$ (complete over/under representation includes a set of $n$ pairs with the integers from 1 to $2n$ appearing exactly once.)

Assign

\[
\begin{align*}
y_1 &= 2 + 1 + t_f \\
y_m &= y_{(m-1)} + 1 + t_m \\
y_n &= 2
\end{align*}
\]

If $y_{c_i} - y_{(c_i-1)} = 2$, then $x_i = y_{c_i} - 1$

If $y_{c_i} - y_{(c_i-1)} = 3$, then if $y_1 + y_{c_i} = \text{even}$, then $x_i = y_{c_i} - 1$, else $x_i = y_{c_i} - 2$

(It has not yet been determined how to handle the case where $y_{c_i} - y_{(c_i-1)} > 3$, so for now, these matrices will not be transformed.)

If $x_i = x_j$ for $i \neq j$, terminate the knot.

If $|x_i - y_i| = 1$, terminate the knot

If $|x_i - y_i| = 3$ and $|y_j - y_k| \neq 1$ for $x_j = \min\{x_i, y_i\} + 1$ and $x_k = \min\{x_i, y_i\} + 2$, terminate the knot.

Once the over/under representation has eliminated many of the options, consider the matrix representations again. The final two steps will eliminate duplicates from column/row equivalence and some TYPE III Reidemeister moves.

Step 3:

If possible, chose a column, $x$, such that $c_r \neq x \quad \forall r$. (Note: Rows $x$ and $x + 1$ are the rows that will change.)

If $c_s = c_y$ for some $y$, then $c'_{s+1} = c_s$ and if $y \neq x + 1$ then $c'_s = y$, else $c'_s = y + 1$

If $c_{x+1} = c_y$ for some $y$, then $c'_{s+1} = c_{x+1}$ and if $y \neq x$ then $c'_{s+1} = y$, else $c'_{s+1} = y + 1$
If another matrix exists with \( c_1 \) through \( c_n \) differing only from the original by \( c'_{i+1} \) and \( c'_i \) in place of \( c_{i+1} \) and \( c_i \), terminate the matrix.

Repeat this step on every matrix until every option is exhausted and on every created matrix until the original matrix is recovered.

Step 4:

If \( c'_{r} = c_{r-1} + 1 \) \( \forall r \) eliminate the matrix

If \( c'_{n-r+1} = n - c_r + 2 \) \( \forall r \) eliminate the matrix

This step eliminates all column/row equivalent duplicates left in matrix construction.

Step 5: Create the matrices.

For each matrix entry from \( a_{1,1} \) to \( a_{n,n} \),

\[
\begin{align*}
  &a_{i,i} = a_{i,(i+1)} = a_{n,1} = a_{n,n} = 1 \quad \forall i < n ; \\
  &a_{1,1} = a_{2,2} = a_{m,n} = -2 \quad \forall m \leq n ; \text{ and} \\
  &a_{i,j} = 0 \quad \text{otherwise}
\end{align*}
\]

Conclusion

Realistically, this new process of knot counting still creates an over-count on the number of unique knots of a given number of crossings. There are known issues that have not been dealt with and more than likely there are issues in knots of larger sizes that have yet to be realized. Thus far, this model at the very least produces accurate results for knots with three, four, and five crossings. Once the algorithm is set into action, more will be revealed about the accuracy and the problems remaining in the process.

A useful result of a preliminary step of this method is the relationship of singular values to knots. It was discovered that row/column equivalent knots have identical singular values. This finding provides a useful link between knot theory and linear algebra.

The main success of this project, however, is found in the creation and improvement of
the knot-counting algorithm. From start to finish, the pre-rule matrix construction over count went from an unrealistic \( (n^2)^3 \) to a much more practical \( (n - 2)^n \) matrices. This improvement is huge and the result is the formation of an algorithm that is actually useful.

Hopefully, this project will be continued in the future. The next step is to set up the algorithm on a computer and get it running. Once the accuracy of the program is evaluated, new instances of over-counts will surface. These problems, along with those already known, will then be investigated to eliminate duplication. In this way, new rules will be created and the algorithm will be modified; little by little, the process will improve. Ultimately, this project has made significant progress toward the development of a new and accurate method of knot counting.
References

