Introducing Topology through Simulations of Knot Tying and Band Forming Activities

Lisa E. Marano
West Chester University of Pennsylvania
Department of Mathematics
West Chester, PA 19383
e-mail address: lmarano@wcupa.edu

I. Introduction.

In the Spring of 2004, I began an independent research project with a pair of undergraduate mathematics education majors. The students, Matthew Davis and Meghan Kelley, and I meet once a week. Being sophomores, they had recently completed three semesters of Calculus and Linear Algebra and were enrolled in Differential Equations, and Modern Geometry. In addition, Matthew had taken a course in Probability and Statistics. At this time, neither student had a course in Point Set Topology or Combinatorics.

The following summary of an article appearing in the November 1997 issue of the College Mathematics Journal provides the motivation for the students' project. Kathy Liebars wrote a Classroom Capsule titled *"Loopers"* detailing a classroom activity designed for students in an introductory probability and statistics course. With students in groups of two, have one student hold three strings in her closed hand. The other student randomly ties pairs of string ends together until every pair has been tied. Before the other student opens her hand, have the students make predictions about what they will see. Students realize that the result will either be one, two or three loops. The student opens her hand and records the number of loops observes. The students repeat the activity and the class tallies the results to get empirical estimates on the probability of getting one, two or three loops. Finally, the students try to determine the theoretical probabilities and compare them with the empirical results obtained in class.

Matthew and Meghan worked to generalize this activity. First, they developed a method to simulate the in-class knot-tying activity described above using n strings. Next they determined the theoretical results for n strings. Finally, to extend the activity further, they decided to change the strings to strips of paper and allow the person "tying the knot" to twist the strip of paper a half turn before attaching it to another random paper end. In this paper, we will discuss the failures and successes that the students experienced. In addition, we discuss how the students incorporated technology in finding the solutions to these problems. Finally, we discuss how the students learned some of the basic concepts developed in a Point Set Topology course via working on this project.

II. Empirical estimates for the probability of getting k loops using n strings.

The students began by repeating the experiment described in [1] with four, five and six strings and computed the theoretical probabilities as well. Table I provides an example of their findings for five strings based on 100 trials as compared to theoretical results based on five strings.

Table I

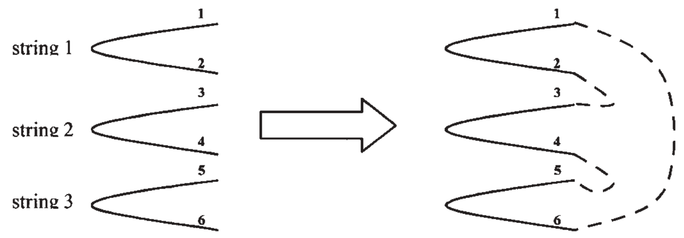
number of loops obtained	1	2	3	4	5
empirical results	0.49	0.38	0.12	0.01	0
theoretical results	0.406	0.423	0.148	0.021	0.001

While performing the trials using four, five and six strings, several questions arose. The students wondered if they should be making a distinction between loops made from just one string versus loops made from two, three or more strings. This became an excellent opportunity to explain the importance of having a well defined problem and to introduce some concepts of elementary topology. In a quest to better define their problem, the students listed similarities and differences between a loop made from *j* strings and a loop made from *k* strings. Obviously, they cited that both were loops and the only difference was in their lengths. They decided that their problem would be to simply count the number of loops and that the difference in length ultimately did not matter. From this little activity, they gained a sense of what it means to topologically equivalent.

Another question that came up was: how could they more affectively simulate this activity? They found through theoretical computations that there were 15 distinct outcomes when three strings are used; but with the addition of two strings, the number of distinct outcomes increases to 995. The students realized that using actual strings was time consuming and not feasible. Thus they decided to write a computer program to simulate the activity. They used Excel 2000 and VBA (Visual Basic for Applications) as their development environment.

Before writing the program, the students outlined how the program would generate a random tying of *n* strings. Initially, they assumed that if they performed a random shuffle of the integers 1 through 2*n*, which represent the 2*n* string ends, the shuffle would produce a random tying of the *n* strings. But a problem occurred when deciding how this shuffle would be interpreted. They decided on the following algorithm: Let *S* be a random shuffle of the digits 1 through 2*n*. So $S = (s_1, s_2, s_3, \ldots, s_n)$. Next, group the elements in the shuffle into pairs of two. Thus s_1 is paired with s_2, s_3 with s_4 and so on. Each pairing will represent a knot tying string end s_{2i-1} with string end s_{2i}. This process is illustrated in Figure 1 below.

Figure 1: Example of Tying using Random Shuffle
Given three strings with ends labeled 1 – 6, consider the following random shuffle of these digits: 2, 3, 5, 4, 1, 6. Grouping consecutive pairs yields: (2,3), (5,4), (1,6). Tie each pair of string ends together.



The students realized early on that this algorithm was not easy to implement, for it was difficult to keep track of which strings were connected and therefore difficult to count how many loops were formed. After altering the method described above, the students developed the modified algorithm described below in Figure 2 to simulate the random knot tying. In addition, Figure 2b demonstrates how this algorithm would be used to randomly tie together 3 strings.

Figure 2a: Modified Algorithm to simulate knot tying

1. Without loss of generality, assume the random shuffle begins with string end 1; call this the "loop seed." Set loop counter to zero.
2. Choose, at random, one of the string ends remaining of 2 through $2n$; call this "next number" and remove "next number" from list.
3. Check to see if "loop seed" and "next number" share the same string.
    a. If so, add one to loop counter and select at random one of the string ends remaining. Call this the new "loop seed".
    b. If not, choose the string end that shares the same string with "next number" and call this "next number". Remove "next number" from list.
4. If there are string ends remaining, repeat process starting at step 2.

Figure 2b: Using algorithm to randomly tie together 3 strings

1. Start with "loop seed" = 1.
2. Randomly select a number from 2 – 6; "next number" = 3; remove 3 from list of available string ends.
3. Are "next number" and "loop seed" from the same string? Nope! So select string end that corresponds to string including "next number". New "next number" = 4 and remove 4 from list of available string ends.
4. There are strings remaining, thus randomly select a number from 2, 5 and 6.
   "next number" = 2.
5. Are "next number" and "loop seed" from the same string? Yes! Increment "loop counter". Randomly select a number from 5, 6. "loop seed" = 5.
6. Chose last remaining number; "next number" = 6.
7. Are "next number" and "loop seed" from the same string? Yes! Increment "loop counter".

They incorporated this algorithm into their VBA program and we include, in Figure 3, a sample of the output generated by their simulation program.

Figure 3: Sample Output

Input			**Output**							
Number of Strings	Number of Trials	Number of repetitions	Number of loops formed	Rep 1	Rep 2	Rep 3	...Rep 39	Rep 40	**Totals**	**Proportion**
10	250	40	1	70	87	60	65	85	2936	0.2936
			2	96	91	106	92	104	3966	0.3966
			3	65	56	54	59	40	2257	0.2257
			4	18	13	25	29	16	707	0.0707
			5	1	3	4	5	5	115	0.0115
			6	0	0	1	0	0	19	0.0019
			7	0	0	0	0	0	0	0
			8	0	0	0	0	0	0	0
			9	0	0	0	0	0	0	0
			10	0	0	0	0	0	0	0

III. Determining Theoretical Results.

Convinced that their algorithm worked, they wanted to see how well it worked compared to theoretical results for large values of n. We continued to meet weekly throughout the summer with the goal of determining the probability distribution for the number of loops created using n strings. The students realized that there was a link between the different ways they could make k loops using n strings and all distinct sequences of positive integers of length k whose sum is n, see Figure 4 for example. This was their first meaningful introduction to mathematical research and how serendipitous it could be.

Figure 4: Example of the Relationship between Loops and Partitions

Given 7 strings, you can create 4 loops by

1. 1 loop of length 4 and 3 loops of length 1,
2. 1 loop of length 3, 1 loop of length 2 and 2 loops of length 1, and
3. 3 loops of length 2 and 1 loop of length 1.

The distinct sequences of positive integers of length 4 whose sum is 7 are:

a. (4, 1, 1, 1), b. (3, 2, 1, 1), and c. (2, 2, 2, 1).

They determined that each distinct sequence of positive integers of length k whose sum is n, $A_j = \left(a_{i,j}\right)_{i=1}^{k}$, contributes N_j different ways to create k loops where N_j is given by the following: $N_j = 2^{n-k}\binom{n}{a_{1,j}}\left(a_{1,j}-1\right)!\binom{n-a_{1,j}}{a_{2,j}}\left(a_{2,j}-1\right)!\binom{n-a_{1,j}-a_{2,j}}{a_{3,j}}\left(a_{3,j}-1\right)!\ldots\binom{n-\sum_{i=1}^{k-1}a_{i,j}}{a_{k,j}}\left(a_{k,j}-1\right)!$

Hence, once they've found all such sequences A_j, they would simply sum the N_j's. Thus ***all*** they had to do was find a closed formula which would generate all such sequences A_j.

Around this same time, Meghan was taking a course in Combinatorics and stumbled upon the section of her text titled Partitions. She immediately noticed the link between what we were working on and what her book described as the number of partitions of n into k parts. She was disappointed to learn that there is no known closed formula for the number of k of partitions of n. They both realized that in order to determine the probability distribution for the number of loops created using n strings, they would have to write yet another program.

Since they already established how many distinct k loops each sequence A_j would create, the real work was to produce an algorithm to find all such sequences. They developed a recursive method of doing so, see Figure 5. This time they used C++ as their programming language. In addition, Figure 6 demonstrates how the algorithm is used to generate all partitions of length 6 whose sum is 10.

Figure 5: Algorithm for determining all non-increasing sequences of length *k* whose sum is *n*.

1. Initialize first sequence by setting all k entries to '1'.
2. If the sum is greater than n, stop. If not, increment the first element until the sum is n. This becomes the "new" first sequence.
3. Look for the first pair of consecutive numbers that differ by more than one.
   a. If a pair exists, increment the second by one and set the first equal to the second. Repeat Step 2.
   b. If a pair does not exist, go to step 4.
4. Find the first '1' on the list, and increment it by one to '2'; and reset all prior numbers in sequence to '2' as well. Repeat Step 2.
5. If there are no '1's left, you're done.

Figure 6: Using the algorithm to determine all partitions of length 6 whose sum is 10.

1. Initialize sequence: 1 1 1 1 1 1
2. Increment first until sum equals 10: **5 1 1 1 1 1**
3. Is there a consecutive pair whose difference is greater than 1? Yes. Re-initialize: 2 2 1 1 1 1
4. Increment first until sum equals 10: **4 2 1 1 1 1**
5. Is there a consecutive pair whose difference is greater than 1? Yes. Re-initialize: 3 3 1 1 1 1
6. Increment first until sum equals 10: **3 3 1 1 1 1**
7. Is there a consecutive pair whose difference is greater than 1? No. Then go to first 1, increase it by one and re-initialize: 2 2 2 1 1 1
8. Increment first until sum equals 10: **3 2 2 1 1 1**
9. Is there a consecutive pair whose difference is greater than 1? No. Then go to first 1, increase it by one and re-initialize: 2 2 2 2 1 1
10. Increment first until sum equals 10: **2 2 2 2 1 1**

IV. Putting a Twist in the Problem.

As mentioned earlier, after generalizing the knot tying problem to *n* strings, the students moved on to strips of papers that would be "tied" together; but before "tying" each pair, a single twist of one end is allowed. In order to redefine the problem in this context, an introduction to several concepts from elementary topology such as manifolds, orientation preserving and orientation reversing paths, orientable vs. nonorientable manifolds, equivalence relations, among others, was needed. After which, the students decided to only make a distinction between one-sided (nonorientable manifolds) and two-sided (orientable) bands. That is to say that all bands with an even number of twists indistinguishable, and similarly, all bands with an odd number of twists were considered indistinguishable.

We found that with just two strips of paper, the calculations are a little messy! That is, with just two strips, you can have: 1) 2 two-sided bands with probability 1/12, 2) 2 one-sided bands with probability 1/12, 3) 1 two-sided and 1 one-sided with probability 2/12, 4) 1 two-sided band with probability 4/12 or 5) 1 one-sided band with probability 4/12.

Although we've added a new *twist* to the original problem, there is still a connection to partitions and thus Matthew and Meghan are working to modify their C++ program to account for the new variation.

V. What next?

After completing this phase of the project, we will work on randomly identifying edges of rectangles. This will certainly to lead to further discussion of topological properties and more programming we're sure!

VI. Bibliography

[1] Liebars, Cathy, "Tying up loose ends in Probability," *The College Mathematics Journal*, Vol. 28, No. 5.