A game on pretzel links

Jarell Cheong Tze Wen

Harvard University, Cambridge MA 02138

Abstract. This expository paper introduces a two-player, zero-sum game to study the structure of pretzel links. A simplified version of the game that can be played in the command-line is made with Python, and basic combinatorial game theory is used to construct an algorithm that makes better-than-random moves. In particular, we find that results in knot theory allow for a computationally trivial detection of game completion.

1 Introduction

1.1 Pretzel links

Pretzel links are a special class of links characterized by the way their intersections are arranged. They are an appealing type of link to study due to the existence of a ”standard projection,” which lets their data be encoded efficiently in a tuple.

To understand pretzel links, we must first understand their building blocks: tangles. While it is possible to define a tangle as the proper embedding of a disjoint union of arcs into a 3-ball [2], it suffices for the purposes of this discussion to stick with the following, much more elementary, description [5].

Definition 1.1. A *tangle of order a* is the intertwining of two circular helices with $|a|$ crossings, turning anti-clockwise if $a > 0$ and clockwise otherwise.

Thus, a tangle can be completely described as an integer, positive if the strand from the top-left corner is below the strand from the top-right corner and negative if the converse is true. Of course, the zero tangle is just two unconnected strands, but this is an important degenerate case to keep in mind.

With the above definition for tangles, we can then turn our attention to the pretzel link proper. Its definition is given as follows [3].

Definition 1.2. Let $a_1, \dots, a_n$ be integers. Then, a *pretzel link $P(a_1, \dots, a_n)$* is the link obtained by joining tangles of orders $a_1, \dots, a_n$ circularly and anti-clockwise.

Note that by join, we mean that for every pair of adjacent tangles, the top-right strand of the left tangle is connected to the top-left strand of the right tangle by an imaginary string. Then, to complete the pretzel link, the top-left strand of the leftmost tangle is connected to the top-right strand of the rightmost tangle. Bottom-left and bottom-right strands are connected analogously.

Now, we proceed by listing some useful properties of pretzel links. The first property in this list harnesses the fact that a series of Reidermeister moves lets us view a pretzel link as two circular loops (one above and one below) along with tangles attached to both loops [1]. Obviously, rotating the loops does not change the ambient pretzel link, so the result below follows.

Theorem 1.3. *Let σ be a cyclic permutation of $a_1, \dots, a_n$. Then, the pretzel links $P(a_1, \dots, a_n)$ and $P(\sigma(a_1, \dots, a_n))$ are isotopic.*

Next, by examining the interactions between two adjacent tangles, both of which have small order, we can deduce more relations within a pretzel link in its standard tuple presentation. Basic Reidermeister moves are all we need to justify the following isotopies.

Theorem 1.4. *Let $a_1, \dots, a_n$ be integers. Then, the following are all isotopies of pretzel links.*

(a) $P(\dots, a_i, \pm 1, \dots) \cong P(\dots, \pm 1, a_i, \dots)$;

(b) $P(\dots, \pm 1, \mp 1, \dots) \cong P(\dots)$;

(c) $P(\dots, \pm 1, \mp 2, \dots) \cong P(\dots, \pm 2, \dots)$;

We also examine the special case when $n = 2$. With a series of Reidermeister moves, we can transform a pretzel link with $n = 2$ into a torus link, a link in another special family of links that have been studied extensively. Moreover, we know when torus links are trivial [6], and this contributes to the theory on completion detection.

Theorem 1.5. *The pretzel link $P(a_1, a_2)$ is just the torus link $T(2, n)$. In particular, this torus link is the unlink if and only if n is 1 or -1.*

1.2 The pretzel game

Finally, we move on to the main discussion of this paper: the pretzel game. The proposed game is two-player and zero-sum, which means that any advantage one player gains is lost in equal magnitude by the other. The underlying structure of the game (the pretzel link) is also easy to comprehend.

The moves of the game, however, are not so trivial. We first need to elaborate on the concept of skein relations, i.e. transformations to the regions L_0, L_-, and L_+ as they are conventionally defined [8]. More formally, we have the following definition.

Definition 1.6. A *simplifying skein relation* is an operation on a link where any L_- or L_+ intersection is reduced to its corresponding L_0 region, otherwise leaving the rest of the link unchanged.

Now, the proposed game can be described with the following story.

Definition 1.7. The *pretzel game* is played as follows. Two players, A and B, take turns performing simplifying skein relations (moves) on a randomly generated pretzel link. The game ends when the given link is made into the unlink; the player that makes the last move wins.

Notice that the list (in Python) is a particularly appealing data structure on which to model our game. Each move either decreases the order of a tangle by 1 or eliminates it completely, and this process can be carried out using lists with relative ease. The problem of detecting when a list represents the unlink is the primary focus of the next section.

Remark. To simplify our game, we enforce the rule that reducing any tangle to the zero tangle is not allowed (so that we may never have 0's in any list). However, if this rule is not enforced, a feasible algorithm can still be constructed, albeit one that is much more complicated.

Indeed, if some a_i is 0, we can use the cyclic symmetry of the pretzel link to assume that $1 < i < n$. Then, P is a connected sum $P_1 \# P_2$, where $P_1 = P(a_1, \ldots, a_{i-1}, 0)$ and $P_2 = P(0, a_{i+1}, \ldots, a_n)$. We can repeat this process, and at its termination, the link is non-trivial if any summand in the connected sum is non-trivial [7].

2 Preliminaries

2.1 Detecting completion

We concern ourselves now with the detection of game completion. In other words, we answer the question: can we write a function that takes in a pretzel link as input and outputs the triviality of said link? It turns out, ideas from knot theory are rather useful in tackling this problem.

First, to clarify matters, we introduce a normal form presentation for any pretzel link. It is the case that every pretzel link can be expressed in this form, when computations with the rules given in the preceding section are simpler.

Definition 2.1. A *normal form presentation* for a pretzel link P is one of the form $P(1, \ldots, 1, a_1, \ldots, a_n)$, where $|a_i| > 1$ for all i and no $a_i = -2$, or one of the form $P(a_1, \ldots, a_n, -1, \ldots, -1)$, where $|a_i| > 1$ for all i and no $a_i = 2$.

Furthermore, by leveraging the mutability and methods of Python lists, it is not too hard to convert any pretzel link (represented as a list) into one in normal form. Indeed, the following process can always be used to perform such a conversion. Note how the properties of pretzel links given in the previous section justify the moving of numbers in the list.

Theorem 2.2. *To rewrite a pretzel link written as a list into its normal form, it suffices to perform the steps below in the given order.*

(a) *Move the* 1*'s to the front of the list.*

(b) *Move the* -1*'s to the back of the list.*

(c) *Move the* -2*'s to the front of the list to the right of the* 1*'s.*

(d) *Move the* 2*'s to the back of the list to the left of the* -1*'s.*

(e) *Cancel all matching* 1*'s and* -1*'s at the respective ends of the list.*

(f) *Cancel* -2*'s with* 1*'s or* 2*'s with* -1*'s.*

We can further address how each step is implemented in code. To move the 1's and -1's is a simple task of popping the desired number and inserting it either at the start or end of the list, respectively. To move the 2's and -2's is analogous, except we first have to obtain the index of the leftmost 1 and rightmost -1.

Cancellation, however, is slightly more complex, and to this end, we use recursion. In the base case where there are no cancellations to be made, our cancellation function simply returns the link. However, if there is even a single possible cancellation, we perform that cancellation before calling our function on this updated link.

At last, to detect completion, one final result about the classification of pretzel links is needed. The following theorem is a restatement of Bonahon's theorem on Montesinos links, which was extensively studied by Heiner Zieschang in his 1984 exposition on knot theory [4].

Theorem 2.3. *Let* P *be a pretzel link in normal form. If* $n > 2$*,* P *is not the unlink.*

Remark. The above result is a small corollary of the classification of Montesinos links in the aforementioned book by Zieschang. In the section on Montesinos links in this text, our pretzel links would be expressed as $K(e; 1/a_1, \dots, 1/a_n)$, where e is the number of 1's (if positive) or the number of -1's (if negative).

Since pretzel links with $n = 1$ are obviously trivial, it follows that the following algorithm can be used to detect completion.

Theorem 2.4. *To check if a pretzel link written as a list is trivial, it suffices to perform the steps below in the given order.*

(a) *Write the list in normal form.*

(b) *If $n = 1$, the list is the unlink.*

(c) *If $n = 2$, the list is the unlink if any entry is 1 or -1.*

(d) *Otherwise, the list is not the unlink.*

2.2 Game construction

Denote the human player as the user and the algorithm that makes moves in response as the AI. If we simply allow the AI to make random moves, the construction of the game is fairly straightforward, but not trivial. It suffices to use the following functions to play the pretzel game.

Check. Based on the theory and algorithm developed in the previous section, a list is converted into normal form and checks are performed by conditioning on the length of the resulting list.

Move. By taking in three parameters, namely the *link*, the *index* of the list, and the *operation* to be performed (this is 0 if we want to delete the entry and 1 if we want to reduce the absolute value of the entry), a valid move is performed.

Valid. With the same three parameters as above, this function makes sure that the given index is in the valid range and that the operation is either 0 or 1. It also checks that if the operation is 1, the number at the chosen index is not ± 1, so as to avoid any zero entries which lead to connect sums.

AI Random. Calling on the *random* function in Python, a random move is made. If the move is not valid, the function is called again, and this recursive procedure continues until a valid move is made.

Generate. Using the random module again, a random list is generated. To make the game computationally feasible, the length of the list ℓ is restricted to $[4, 10]$, and the entries $\ell(i) \neq 0$ are restricted to $[-9, 9]$.

Game. A link is generated, and an infinite loop is initialized. Both players take turns to make moves, and the list is checked for triviality after each turn. When a player wins, the loop is broken, and the winner is announced.

However, with the game in its simplified form, it is not too hard to deduce a strategy that is better than pure randomness. Indeed, notice that any list of length 2 should be reduced to one of length 1, and any list of length 3 should have its length preserved. These ideas motivate the following algorithm.

Theorem 2.5. *Let P be a pretzel link. Then, better-than-random play ensues when type* 0 *simplifying skein relations are applied to $P(a_1, a_2)$ links and type* 1 *simplyfing skein relations are applied to $P(a_1, a_2, a_3)$ links.*

Naturally, it follows that an **AI Better** function can be created that makes the better moves than the random function given any link. Lists of length 2 are reduced via operation 0 moves, while lists of length 3 are maintained as such via operation 1 moves. For lists of length greater than 3, the random function is applied for simplicity.

The code required to try out the game in the command-line can be found on GitHub here. Note that the only non-trivial requirement needed to run this game is Python 3, and this can be installed without too much difficulty. As an alternative, a video demonstration of the project can be found here.

References

[1] C. C. Adams. The knot book. *American Mathematical Society*, 1994.

[2] J. H. Conway. An enumeration of knots and links. *Elsevier*, 1970.

[3] R. Ferreol. Pretzel knots and links. *Mathcurve: Pretzel Links*, 2018.

[4] B. Gerhard and H. Zieschang. Knots. *de Gruyter Studies in Mathematics*, 2003.

[5] L. H. Kauffman. On the classification of rational tangles. *Elsevier*, 2004.

[6] K. Murasugi. Knot theory and its applications. *Springer Science*, 2007.

[7] D. Rolfsen. Knots and links. *American Mathematical Society*, 2003.

[8] E. W. Weisstein. Skein relationship. *MathWorld - A Wolfram Web Resource*, 1999.