VISUALIZING THE MANHATTAN CURVE — JOURNAL

WILLIAM CLAMPITT AND GIUSEPPE MARTONE

Abstract. This notebook will serve as a research journal for William's master thesis project

Contents

1. November 19, 2024

1.1. Meeting notes.

1.1.1. *Counting Problems.* Gauss estimated that the distribution of prime numbers was

$$\#\{p \in P\colon p \leq T\} \sim \frac{T}{\ln T}$$

These type of counting problems are common in fields such as Geometry, Topology, and Dynamical Systems.

Typically, in this context

$$\#\{a \in A\colon a \leq T\} \sim \text{ exponential in } T$$

Topoligical Entropy. Typically, topological entropy is calcualted with the below formula:

$$h_{\text{top}}(A) = \lim_{T\to\infty} \frac{1}{T} \ln \#\{a \in A\colon a \leq T\}$$

Example.

$$h_{\text{top}}(\mathbb{N}) = 0$$

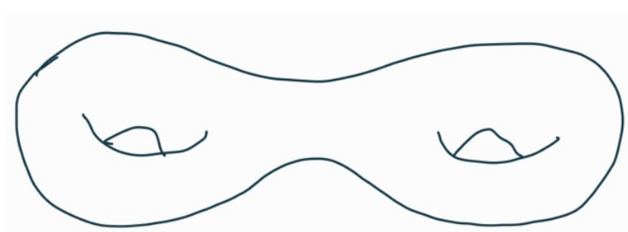

Figure 1. Hyperbolic structure (locally looks like $\mathbb{H}^2$)

Hilbert Metric First Glance.

(1) You need a hyperbolic structure on S, call it p.

Date: November 2024.

(2) The set of closed loops (discrete)
- Each closed loop has a length w.r.t. the hyperbolic structure (number > 0)

We will denote the length of the curve c as $l_p(c)$ w.r.t. $p \in \mathbb{R}$ where $p > 0$.

$$\#\{c \colon l_p(c) < T\} \sim \frac{e^T}{T}$$

Remark 1. This was shown by Huber Marpulis [William] Is this person the same as Grigory Margulis? Maybe I wrote the name down wrong when I was taking notes.

Origin of Program Matrices. The three matrices in the first program are representing a relfection accross the three distinct edges of the triangles that are formed when we stretch the holes of our pants to the boundary of our hyperbolic space.

1.2. **Post-meeting notes.**

1.2.1. *Counting Problems.*

Example. The distribution of prime numbers. In 1792 Gauss proposed that

$$\pi(n) \sim \frac{n}{\ln n}$$

but was later refined to

$$\pi(n) \sim \mathrm{Li}(n),$$

where

$$\mathrm{Li}(n) = \int_2^n \frac{\mathrm{d}x}{\ln x}$$

[1]

2. December 3, 2024

2.1. **Meeting notes.**

Summary:

In this meeting we discussed the problems that I encountered with my program. In summary, my program is using up too much system memory of the computer. This results in the program crashing which prematurely halts the calculation of the words of our alphabet. Because I was not incrementally storing the already computed matrices, the program would not yeild any results if it crashed.

Potential Solutions to Problem. I think one way that might be good to reduce the system memory useage of my program would be to store the words of length n in a file, then use those words to generate the words of length $n+1$. The words of length n would then be unloaded from system memory. After the words of length n are unloaded, we can use the words of length $n+1$ to calculated the words of length $n+2$ and so on.

2.2. **Post-meeting notes.**

I am attempting to rewrite my current program using Octave while also implimenting my idea about iteratively saving the words of length n each time so that I do not have to store the entirety of the list in system memory at one time. After this is done, the list will need to be sifted through to remove duplicates matrices.

So far it seems like the success of this project would be greatly benifited by reducing the number of matrix multiplications a program would have to do. Currently before any duplicates are removed, generating all of the words up to length n creates $3^{\frac{n(n+1)}{2}}$ matrices. This takes a very long time to compute. My theory though is that this creates a lot of duplicates, especially when we are relitavely close to the origin of our tiling. It would be ideal if we could recognize the conditions that would create a duplicate matrix before having to preform the computation.

From what I understand of what Dr. Martone explained to me in one of our previous meetings is that these matrices represent a reflection across one of the edges of our triangles that are formed by our pairs of pants. I am assuming that the tiling process would begin with a single triangle and then continuously reflect across every edge of each triangle that was formed during this process.

I am pretty sure that each word we create is representative of a sort of path formed by the reflections about each edge of these triangles. Many of these paths will end up leading to the same place though. [William] I will provide an illustration of my idea here later. If we could find a more efficient way of tiling the space that minimizes the number of paths that lead to the same place in our tiling, then the computation would hopefully become a lot less resource intensive, which would make it more practical.

I suspect that this could be done by using some sort of graph to map out triangular grid that is formed by our tiling. We could hopefully calculate the size of our graph by finding the maximum "radius" that can be reached by words of length n, and then using some algorithm to find the shortest amount of reflections that would be required to reach that tile. This would probably be a good place to impliment something like Dijkstra's algorithm which finds the shortest path between nodes in a graph. All of this should be able to be done much more efficiently that just brute forcing thousands of matrix multitplications and then sorting out the duplicates.

TODO List:

- ☐ Make sure that I am thinking about these matrices in the right way.
- ☐ If I am thinking about this the right way, figure out how we could efficiently encode this tiling as a graph.
- ☐ I would possibly need to prove that two words that lead to the same triangle actually end up being the same matrix.

References

[1] *Prime Number Theorem*. URL: https://mathworld.wolfram.com/PrimeNumberTheorem.html.