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
1.1. Meeting notes 1
1.2. Post-meeting notes 2
2. December 3, 2024 3
2.1. Meeting notes 3
2.2. Post-meeting notes 3
3. December 14, 2024 3
3.1. Meeting Notes 3
3.2. Post-meeting Notes 4
4. January 9, 2025 4
5. Meeting Notes 4
6. January 14, 2025 5
6.1. Meeting Notes 5
The General Case for Distance 5
6.2. Post-Meeting Notes 5
7. January 17, 2025 5
7.1. Meeting Notes 5
7.2. Post-Meeting Notes 5
8. January 24, 2025 5
8.1. Meeting Notes 5
8.2. Post-Meeting Notes 5
9. Program Explanation 5

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

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

Date: November 2024.

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.

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

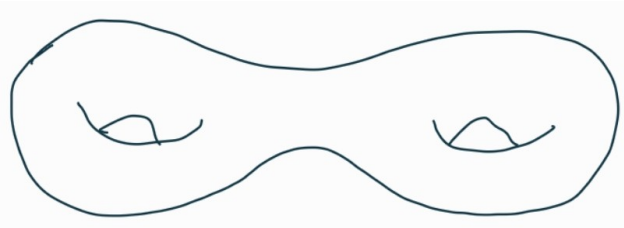

FIGURE 1. Hyperbolic structure (locally looks like $\mathbb{H}^2$)

Hilbert Metric First Glance.

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

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

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

where

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

[wolfram_prime_num_theorem]

[1] G: Huber proved it for hyperbolic surfaces. G. Margulis generalized it and his proof is more dynamical

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.

3. December 14, 2024

3.1. Meeting Notes.

Dr. Martone seems to think that my graph idea is interesting and it worth looking into further. He was concerned about my graph needing to find the shortest Hilbert distance between nodes; not the shortest number of reflections between nodes. My initial though was that I could weight the edges

of he graph in such a way that they mimicked the Hilbert distance. This may still be possible, but it would require more thought to get working properly.

3.2. **Post-meeting Notes.** I have spent some time working on this program. My first step is to find a way of encoding the graph of words up to a maximum length of N as an adjacency matrix that the computer can work with. I made some observations about the number of triangles that are added to the tiling when going from a max of N words to a max of $N+1$ words.

length of words	number of nodes added	sum total of nodes
0	1	1
1	3	4
2	6	10
3	9	19
4	12	31

This can be extrapolated into the formula

$$\#\text{ Nodes}(N) = 1 + \sum_{n=1}^{N} 3n = 1 + \frac{3N(N+1)}{2}$$

From the work that I have done so far, I think that the most difficult challenge that I am facing right now is encoding the tiling that we have as an adjacency matrix that the computer can work with. Once we have that, there are many implementations of Dijkstra's algorithm that can either be borrowed or modified to fit our needs. I am currently doing some experimentation with algorithms that will generate the adjacency matrix for a graph that represents a tiling of words up to length N.

4. January 9, 2025

5. Meeting Notes

During this meeting we discussed potential ways of weighting the graph that we worked on last time. After some thinking, Giuseppe thought of an idea to use the cross ratio to find weights for our graph. I have a drawing of the setup we came up with in Figure 2.

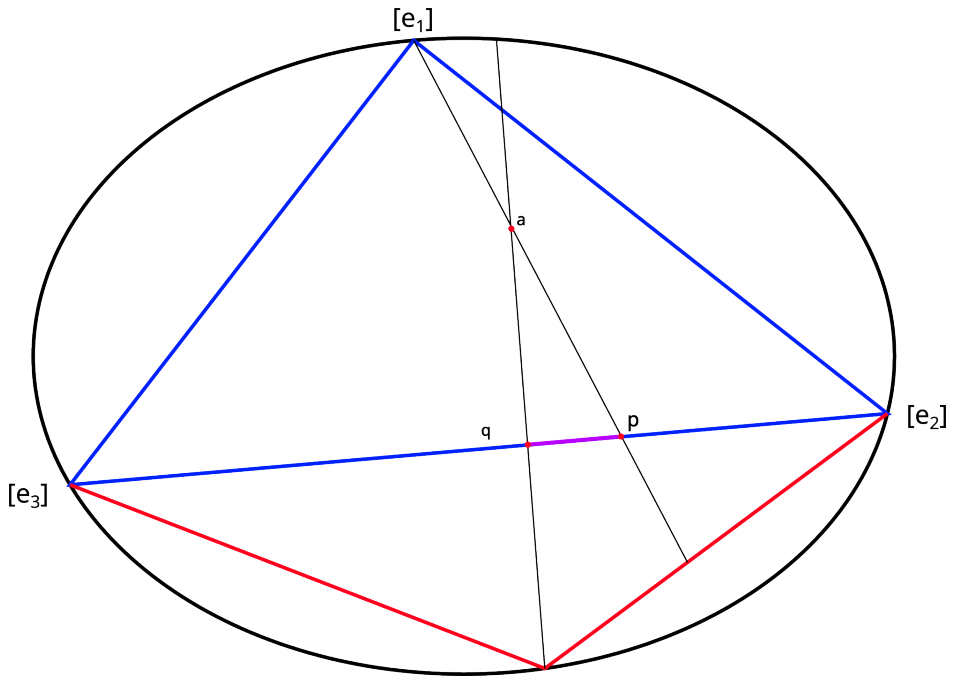

Figure 2. Visualization of method of producing graph weights

6. January 14, 2025

6.1. **Meeting Notes.** I had some trouble figuring out the distance discussed in Figure 2. During our meeting today I realized that for some reason I thought that $[e_1], [e_2], [e_3]$ were our reflection matrices, but really they are the standard basis vectors in $\mathbb{R}^3$. After that was sorted, we were able to calculate the length in the case of R_1.

The General Case for Distance. Since v_2 is the plane spanned by v_1 and v_4, $v_2 = a_1v_1 + a_4v_4$.

Similarly, since v_3 is the plane spanned by v_1 and v_4, $v_3 = b_1v_1 + b_4v_4$.

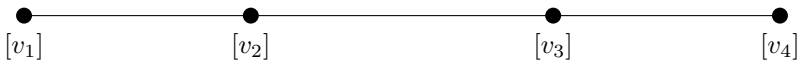

Figure 3. General setup for distance.

The distance between $[v_2]$ and $[v_3]$ would be given by

$$d([v_2], [v_3]) = \ln\left(\mathrm{Cr}\left[v_1, v_2, v_3, v_4\right]\right)$$

The cross ratio would be given by

$$\mathrm{Cr}\left[v_1, v_2, v_3, v_4\right] = \frac{b_4}{a_4} \cdot \frac{a_1}{b_1}$$

Claims:

- $\mathrm{Cr}\left[v_1, v_2, v_3, v_4\right] > 1$
- It doesn't depend on the choice of coordinates. In other words, if we rescale v_1, v_2, v_3, v_4 independently, the cross ratio does not change.
- For any invertible 3×3 matrix A, we have $\mathrm{Cr}\left[Av_1, Av_2, Av_3, Av_4\right] = \mathrm{Cr}\left[v_1, v_2, v_3, v_4\right]$.

6.2. **Post-Meeting Notes.** I was able to calculate the other two distances on my own after our meeting last Tuesday. From my calculations, each edge of our graph for R_1, R_2, and R_3 should be weighted as $\ln\left(\frac{t+2}{2t^2+t}\right)$

7. January 17, 2025

7.1. **Meeting Notes.** During this meeting, we discovered that our method of tiling was not accurate to what is actually going on. Instead of a regular graph that can connect back on itself, we really have more of a tree structure.

7.2. **Post-Meeting Notes.**

8. January 24, 2025

8.1. **Meeting Notes.**

8.2. **Post-Meeting Notes.**

9. Program Explanation

After we discovered that our tiling was more reminiscent of a tree structure, I rewrote the code to generate an adjacency matrix that reflects this tree. At its current stage, my adjacency matrix only represents one of the main branches of the tree and generates N layers of nodes.

After the adjacency matrix is generated, the program encodes the edges of the tree with the reflections that they represent. A 1 is used for R_1, a 2 for R_2, and a 3 for R_3.

Insert picture of tree here

Figure 4. Tree structure created by the reflections of starting triangle.

LISTING 1. Code that generates the adjacency matrix for the tree.

```
function adj_matrix = make_adjacency_matrix(N)

adj_matrix = [0];

if N <= 1
	return;
endif

% Creates a vector with the total number of nodes after generating a tree with depth N
N_k = [[1], zeros(1, N-1)];
for k = [2:N]
	N_k(k) = N_k(k-1) + 2^(k - 1);
endfor

% Makes a blank adjacency matrix and then connects the nodes with 1's
adj_matrix = zeros(N_k(end));

col = 2;
for row = [1:N_k(end-1)]
	adj_matrix(row, col) = 1;
	adj_matrix(row, col+1) = 1;
	col = col + 2;
endfor

% Encodes the reflections into the matrix using 1, 2, and 3 to represent R1, R2, and R3,
    ↪ respectively.
adj_matrix = encode_reflections(adj_matrix, N_k, 1);
adj_matrix = adj_matrix + adj_matrix';

% Replaces 0's with inf
adj_matrix(adj_matrix == 0) = inf;

endfunction

function adj_matrix = encode_reflections(A, N_k, starting_reflection)
	patterns = cell(3, 2);
	patterns(1,:) = {[3, 2, 1], [1, 2, 3]};

	for i = [1:N_k(end - 1)]
		col_count = 0;
		for j = [1:columns(A)]
			if col_count >= 2
				break;
			endif
			if A(i,j) == 1
				switch(mod(j, 6))
					case {0, 2}
						A(i,j) = 3;
					case {3, 5}
						A(i,j) = 2;
					case {1, 4}
						A(i,j) = 1;
				endswitch
```

```
                            col_count = col_count + 1;
                    endif
                endfor

        endfor

        adj_matrix = A;
endfunction
```

I then use Dijkstra's algorithm to find the the path between the root node and any other node in branches below it. My current implementation uses redundant calculations and could be further optimized. The implementation I used for Dijkstra's algorithm can be found at Fix citation here (Works in overleaf but not texstudio)[**alexey_dijkstra_alg**].

After it generates a path from the root node to another node in the tree, my program multiplies the matrices together that would form the path from the root node to the node that we are currently working with. Once this is done, we find the singular values of the matrix and assign the "length" of that matrix to be $l = \ln\left(\frac{\sigma_3}{\sigma_1}\right)$. The program then looks at the value of l and adds a tally to a list that keeps track of how many of these l values fall in the interval $[n, n+1)$.

LISTING 2. Script that combines all of the steps together.

```
N = 10;
t = 1;


disp("Generating␣adjacency␣matrix")
adj_matrix = make_adjacency_matrix(N);
disp("Generating␣adjacency␣matrix:␣finished");

% Possible future optimization: Calculate the length of each step in a path as
% the loop progresses so that it doesn't have to recalculate the path for every node

ls = zeros(1, columns(adj_matrix));

% For right now, I am only looking at the branch where the starting reflection is R1


disp("Converting␣paths␣to␣matrices");
for i = [2:columns(adj_matrix)]
        % P is the path from node 1 to node i
        P = dijkstra_alg(adj_matrix, 1, i);
        M = path_to_mat(adj_matrix, P, t);
        ls(i) = L_t(M);
endfor
disp("Converting␣paths␣to␣matrices:␣finished")

disp("Grouping␣lengths");
groups = group_buckets(ls);
disp("Grouping␣lengths:␣finshed")

xs = groups(1,:);
ys = groups(2,:);

bar(xs, ys);
```