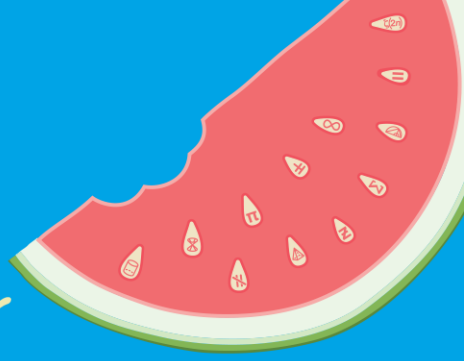
AMSI VACATION RESEARCH SCHOLARSHIPS 2021–22

Get a taste for Research this Summer



Towards a Uniform Sampling Procedure for Abstract Triangulations of Surfaces

Rajan Shankar

Supervised by Dr Jonathan Spreer

The University of Sydney

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Abstract

The study of surfaces is central to many areas of mathematics, both pure and applied. Triangulations are a way of decomposing surfaces into triangles. Here, we consider combinatorial isomorphism types of triangulations. For a variety of purposes, it is of interest to understand what the "average" such isomorphism type of a triangulation looks like. A uniform sampling procedure for triangulations provides a way of doing so. In this report, we describe and deal with the issues that arise with developing such a procedure, and implement the procedure into code. We then collect and visualise data based on many samples of triangulations. One finding is that, for a pre-specified number of triangles $2n$, most of the probability mass of the genus distribution sits within two units of the mean. Another finding is that the average number of symmetries decays exponentially as n increases.

1 Introduction

Consider a computer game with 3-dimensional graphics. Such a game requires a terrain-rendering engine to produce the beautiful landscapes that a player can explore and interact with. Or, consider the problem of visualising 3-dimensional data sourced from an MRI scan of a patient's brain. These problems call for an efficient way of approximating 3-dimensional reality, which motivates our study of surfaces; an object that is central to many areas of mathematics, both pure and applied.

One way to approximate a surface is via a discrete, mathematical object known as a triangulation. This can be thought of as a way of "pixelating" a surface into triangles. Once this is achieved, algorithms for computing things of interest on surfaces may be run.

The problem is that, run-time analyses conducted on these algorithms are generally done with a worst-case triangulation in mind, effectively giving an upper bound on the run-time. This upper bound may vastly overestimate the expected run-time of the algorithm on average-case triangulations. The only way to understand what the average-case triangulation looks like is to develop a uniform sampling procedure for triangulations.

It is important to note that these triangulations that we speak of do not possess tangible coordinates; they are *abstract*, and so are not embedded in a space. See Definition 2.2 for further clarification. Thus, these triangulations that we wish to sample are actually *combinatorial isomorphism types*.

We do not sample triangulations directly; in Section 2 of the report, we describe how a triangulation can be decomposed into a graph-encoded manifold (GEM) and then into a pair of permutations. It is the pairs of permutations that are the objects we sample.

However, there is not a one-to-one correspondence between a triangulation and a pair of permutations. In Section 3, we explain how to deal with this issue by detecting the occurrence of disconnected GEMs and by introducing the idea of weights based on the number of isomorphisms and symmetries. In particular, multiple pairs of permutations may correspond to the "same" GEM. These GEMs have a higher chance of being sampled, thus biasing our uniform sample.

Finally, in Section 4, we implement our uniform sampling procedure in Python and collect data on the

triangulations that we sample. We visualise the data and comment on any interesting behaviour that arises in the plots.

Statement of Authorship. Rajan Shankar implemented the uniform sampling procedure in Python, produced experimental results via simulations, and composed this report. Jonathan Spreer conceived the mathematical theory behind the uniform sampling procedure, provided support and supervision, and proofread this report.

2 Background

We begin by defining the notion of a *surface*.

Definition 2.1 (Surface)**.** A *surface* is a space which locally resembles $\mathbb{R}^2$. That is, every point in such a space has a neighbourhood that is homeomorphic to $\mathbb{R}^2$.

We restrict our research to surfaces that are both *closed* and *orientable*. A closed surface does not have a boundary; informally, this can be thought of as one that cannot be fallen off of, such as the sphere. An orientable surface can be thought of as a one where it is not possible to end up on the other side of the surface by simply walking along it. The Möbius strip is the classic example of a non-orientable surface.

These restrictions allow us to (1) classify a surface by its genus, and (2) ensure that the graph-encoded manifold of a triangulation of a surface is bipartite. While gibberish at the moment, these points will become clearer as we introduce more definitions.

Surfaces are very intricate objects that are difficult to process by a computer. This motivates the notion of an *abstract triangulations* of a surface; a way of decomposing a surface into a more digestible form.

Definition 2.2 (Triangulation)**.** A *(abstract) triangulation* is a decomposition of a surface into $2n$ triangles, $n \in \mathbb{Z}^+$, such that the $3 \times 2n = 6n$ edges of the triangles are glued together pairwise. *Abstract* here simply means that the triangulation is not embedded in a space, i.e. the vertices of the triangulation do not possess coordinates.

It is important to note that triangulations are not the only tool used to study surfaces; approaches from algebraic geometry are used as well. However, the discrete nature of triangulations opens up the study of surfaces to combinatorics and algorithms, which can then be easily handled by a computer.

To work with a triangulation, we would like to first transform it into a *graph-encoded manifold*.

Definition 2.3 (Graph-encoded manifold (GEM))**.** A *graph-encoded manifold* (G, C) is a graph G with an arc colouring C that uniquely encodes a triangulation through the following procedure:

1. Colour the vertices of the triangulation using three colours (red, green and blue by convention) such that each triangle has a different colour on each vertex.

2. Treating each triangle as a node, place an arc between each pair of triangles that share a common edge. Colour each arc according to the single colour that was not used by the two vertices adjacent to the edge that the arc crosses.

3. The nodes and the coloured arcs form the GEM. Rearrange the GEM into a "standard form", where the nodes are separated into two sets with vertical red arcs joining nodes between the two sets.

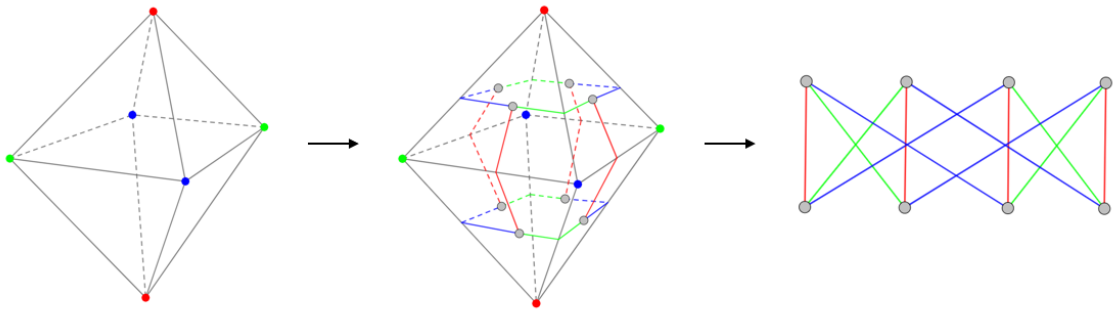


Figure 2.1: An example of obtaining the GEM from an octahedron, which is one of the triangulations of a sphere when $n = 4$. First, the vertices of the triangulation are coloured. Then, coloured arcs are introduced to connect the triangles together. Finally, the arcs are rearranged into the "standard form". See Definition 2.3 for more details.

As already exhibited, we reserve the terms "vertex" and "edge" to be used when discussing triangulations, and the terms "node" and "arc" when discussing GEMs. This is to avoid confusion.

Now, in step 3 of obtaining the GEM from a triangulation, we implicitly assumed that the nodes could be separated into two sets. This assumption is valid due to the *bipartite* nature of the GEM.

Definition 2.4 (Bipartite)**.** A graph is *bipartite* if its nodes can be divided into two sets N_1 and N_2 such that every arc connects a node in N_1 to a node in N_2, and that there are no arcs between nodes of the same set.

Furthermore, this bipartite property is guaranteed through the following proposition.

Proposition 2.1 (Bipartiteness of GEMs (Cavicchioli et al., 1980))**.** A GEM is bipartite if and only if the surface it represents is orientable.

This bipartite property is important because it allows us to use a pair of *permutations* (μ, σ) to represent a GEM.

Definition 2.5 (Permutation, cycle notation)**.** A *permutation* is a bijection $\sigma : \{0, 1, \ldots, n\} \to \{0, 1, \ldots, n\}$, where $n \in \mathbb{Z}^+$. *Cycle notation* is a compact way of writing out a permutation that emphasises the permutation's cycle structure. Each set of brackets represents a cycle. See Figure 2.2 for examples.

By labelling the nodes in a GEM, we can think of the red, green and blue arcs as three different permutations, where N_1 is the domain and N_2 is the codomain of each permutation. By convention, we use μ to denote the permutation represented by the green arcs, and σ for the blue arcs. Note that there is no need for a third permutation for the red arcs as this permutation is always the identity, by construction of our "standard form" of a GEM. Figure 2.3 shows an example of a GEM and a corresponding pair of permutations.

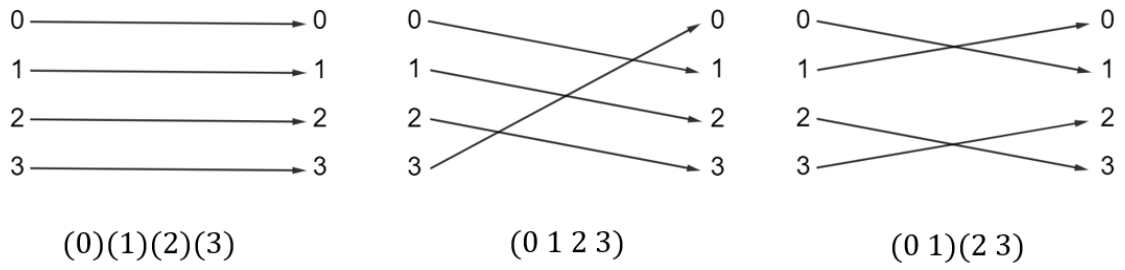


Figure 2.2: Three different permutations and their corresponding cycle notation. On the left is the identity permutation; it has four cycles of length 1. In the middle is a circular-shift permutation; it has one cycle of length 4. On the right is a permutation with two cycles of length 2.

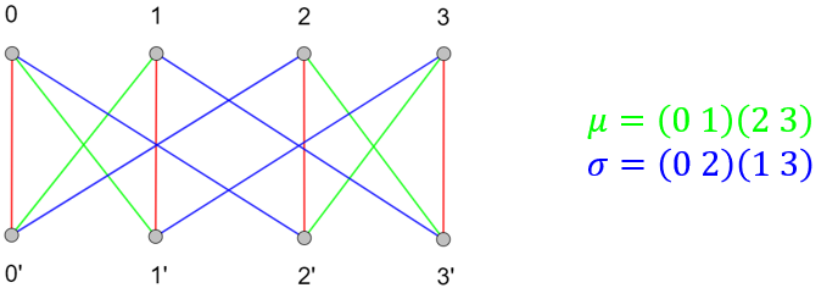


Figure 2.3: Converting a GEM into a pair of permutations. The domain of the permutations consists of the nodes in the top row, which are labelled $0, 1, 2, 3$. The codomain of the permutations consists of the nodes in the bottom row, which are labelled $0', 1', 2', 3'$. μ is the permutation represented by the green-coloured arcs while σ is represented by the blue-coloured arcs.

To recap, we have so far boiled down triangulations to permutations: triangulation $>$ GEM $>$ pair of permutations. This allows us to sample triangulations by sampling pairs of permutations, which is easy to do since permutations are simply ways of shuffling numbers. However, while there is a one-to-one correspondence between triangulations and GEMs, this is NOT the case with GEMs and pairs of permutations. This is what makes it difficult to develop a *uniform* sampling procedure for triangulations, and dealing with this problem is one of the focuses of the next section.

3 Sampling

As briefly stated in the previous section, there is not a one-to-one correspondence between GEMs and pairs of permutations, which makes it difficult to develop a uniform sampling procedure for triangulations. Our goal in this section is to outline such a sampling procedure while explaining any problems we face, or any optimisations

we make, on the way.

First, however, what is the point of coming up with such a sampling procedure? What benefit does it provide to topologists and other researchers who work with surfaces? In short, there exist computer algorithms that solve problems on triangulations. Worst-case run-times of these algorithms are computed based on specially "bad" (read: pathological) triangulations that may be tremendously unrepresentative of what the "average" triangulation looks like. This is a concern because this worst-case run-time may vastly overestimate the average run-time. See for example, Spielman and Teng (2009) on explaining the behavior of algorithms in practice. Having a uniform sampling procedure would provide us with a much better idea on the expected run-times of these algorithms.

We now present the full outline of our uniform sampling procedure. In the subsections that follow, we explain the reasoning behind the key steps of the procedure, as well as provide background and definitions of unfamiliar terminology.

Algorithm 1 Uniform sampling procedure for triangulations

1. Uniformly sample a partition λ of n
   (a) Then, convert λ into its canonical permutation μ_λ
2. Uniformly sample a permutation σ from S_n
3. Check if the subgroup generated by (μ_λ, σ) is transitive
   (a) If not transitive, discard (μ_λ, σ) and go back to step 1
4. Compute GEM
5. Compute symmetries and thus the weight
6. Compute isomorphism signature
7. Add isomorphism signature to the sample with the computed weight

3.1 Integer partitions and canonical permutations

Recall that the number of triangles in a triangulation is given by $2n$, where we have control over the value of n. It is clear why the number of triangles must be even due to the bipartite nature of GEMs. For a given n, the complete set of permutations on n objects is denoted by S_n, also known as the *symmetric group*—a well-studied object in abstract algebra. This space is what we uniformly sample the permutation σ from in step 2 of Algorithm 1.

In step 1, however, we do not obtain our other permutation μ in this way; we instead uniformly sample μ from the set of *canonical permutations* of all possible *cycle structures*.

Definition 3.1 (Canonical permutation of a cycle structure)**.** Let $\sigma \in S_n$. Let a_1 be the number of cycles of

length 1 in σ, a_2 be the number of cycles of length 2, and so on up to a_k being the number of cycles of length k, where $1 \leq k \leq n$. This, we refer to as the *cycle structure* of σ. We say that σ is a *canonical permutation* if it is of the following form:

$$\underbrace{(0\ 1\ \ldots\ k-1)\ldots}_{a_k \text{ cycles}}\underbrace{(ka_k\ ka_k+1\ \ldots\ ka_k+k-2)\ldots}_{a_{k-1} \text{ cycles}}\ \ldots\ \underbrace{\ldots(n)}_{a_1 \text{ cycles}}.$$

In other words, we can obtain the canonical permutation of a given cycle structure by simply taking the numbers $0\ 1\ \ldots\ n$ and inserting brackets appropriately to enforce the cycle structure.

The nice thing about cycle structures is that they can be encoded as *integer partitions*. This allows us to uniformly sample from the set of canonical permutations of cycle structures by uniformly sampling from the set of integer partitions of n, and then converting the resulting partition back into a cycle structure and thus a canonical permutation.

Definition 3.2 (Integer partition)**.** A *partition* λ of an integer n is a way of writing out n as a sum of positive integers. Swapping the order of the summands does not result in a new partition. As an example, the integer 3 can be partitioned in only the following ways: $1+1+1,\ 2+1,\ 3$.

Given an integer partition λ, we denote the canonical permutation based on the cycle structure encoded by λ as μ_λ.

The reason why we prefer sampling from the set of cycle structures instead of from S_n is that it reduces our sample space drastically. Instead of there being $n!$ candidates for μ, we now have roughly $e^{\sqrt{n}}$ candidates, where this approximation comes from Andrews (1977). This reduction is *huge*; even for $n = 10$, we have that $10! = 3,628,800$ whereas $e^{\sqrt{10}} \approx 24$.

As previously stated, we sample cycle structures by sampling integer partitions. However, uniformly sampling from the set of integer partitions of n is not a trivial problem to solve. Fristedt (1993) demonstrate a method to achieve this with an expected time complexity of $O(n^{3/4})$, which is sub-linear. We outline the method below.

Let $X_1, \ldots, X_n$ be n independent geometrically-distributed random variables where for $1 \leq i \leq n$, X_i represents the number of i's in the partition. Also, let p_i be the success probability parameter for X_i, where p_i is given by the following magical number:

$$p_i = 1 - e^{\frac{-\pi}{\sqrt{6n}}i}.$$

We consider X_i to have the support $[0, \infty)$; note that 0 is included in this. Thus, the probability mass functions of our random variables are given by

$$P(X_i = k) = (1 - p_i)^k p_i.$$

Now, we sample $X_1, \ldots, X_n$ according to their probability mass functions. After sampling, we check if the partition they represent sums up to n:

$$\sum_{i=1}^{n} X_i \cdot i \stackrel{?}{=} n.$$

If this is true, then we have a valid partition. If not, we must re-sample $X_1, \ldots, X_n$ and check again. After obtaining a valid partition, the final step is to simply convert it into a permutation with such a cycle structure.

With all this talk about random variables, integer partitions, cycle structures, and canonical permutations, it is useful to finish this subsection with an example that illustrates the sampling workflow for μ_λ.

Example 3.1 (Sampled partition to canonical permutation)**.** Let $n = 7$ and our sample be $X_1 = 3, X_2 = 0, X_3 = 0, X_4 = 1, X_5 = 0, X_6 = 0, X_7 = 0$. It can easily be checked that the sample represents a valid partition (i.e. sums to 7), namely $\lambda = 4 + 1 + 1 + 1$. The canonical permutation μ_λ will have one cycle of length 4 and three cycles of length 1. To get this permutation, we simply take the numbers 0 1 2 3 4 5 6 and insert brackets appropriately to enforce the cycle structure:

$$\mu_\lambda = (0\ 1\ 2\ 3)(4)(5)(6).$$

3.2 Disconnected GEMs

Not every pair of permutations (μ_λ, σ) corresponds to a valid GEM. In some cases, we end up with a *disconnected* GEM; that is, a GEM where it is not possible to reach a node from another node by travelling along its arcs. In a valid (read: connected) GEM, every node should be reachable from every node. Figure 3.1 illustrates an example of a pair of permutations that results in a disconnected GEM.

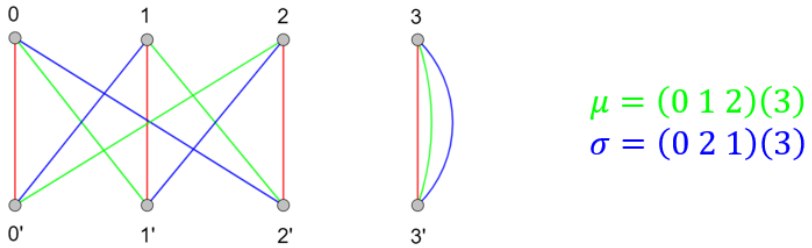


Figure 3.1: An example of a pair of permutations that results in a disconnected GEM.

Fortunately, this problem can be detected by checking if the subgroup generated by (μ_λ, σ) is not *transitive*. This is what we do in step 3 of Algorithm 1. For more information about group actions and transitivity, see the subsection titled 'Abstract Symmetry: Group Operations' in Artin (2010)'s textbook. The following proposition captures the relationship between disconnected GEMs and transitive subgroups.

Proposition 3.1 (Disconnected GEMs and transitive subgroups)**.** A GEM based on the pair of permutations (μ_λ, σ) is disconnected if and only if the subgroup generated by (μ_λ, σ) is not transitive.

Proof. This proposition is simply a consequence of the transitivity property and how GEMs are organised. If a GEM is connected, then without loss of generality, the node labelled 0 can be transported to any other node by compositions of elements of the subgroup. If a subgroup is transitive, then without loss of generality, the number 0 can be mapped to any other number by transporting it across the GEM's arcs to the appropriate node. □

The reason we want to use subgroup transitivity as a proxy for determining whether a GEM is disconnected or not is that it easy to check for transitivity on a computer; for instance, via the `PermutationGroup.is_transitive()` method from the `sympy` Python library.

3.3 Isomorphisms and symmetries

In this subsection, we tackle the opposite problem: multiple pairs of permutations (μ_λ, σ) may correspond to the "same" GEM. This immediately means that uniformly sampling permutations *will not* result in a uniform sampling procedure for GEMs, as GEMs with more pairs of permutations corresponding to them will crop up more often, hence biasing the sample. The approach we take to deal with this problem is to come up with a weight function $1/w$ that takes μ_λ and σ as inputs, and outputs the reciprocal of the number of permutation pairs that correspond to the same GEM. This process is what is outlined in step 5 of Algorithm 1.

First, recall the "standard form" of a GEM being one where the red-coloured arcs were positioned vertically (representing the identity permutation). Now, since μ_λ is sampled from a particular set—the set of cycle structures, we can impose a constraint on the green-coloured arcs too to introduce the notion of the *extended standard form*.

Definition 3.3 (Extended standard form)**.** A GEM is in *extended standard form* if, once its nodes are labelled $0, \ldots, n-1$ and $0', \ldots, (n-1)'$, not only do the red-coloured arcs represent the identity permutation, but also the green-coloured arcs represent μ_λ.

The only "degree of freedom" left is the permutation σ. This is useful in defining *symmetries*. Although first, we introduce *isomorphisms*, which acts as a superset of symmetries.

Definition 3.4 (Isomorphism)**.** Two pairs of permutations (μ_λ, σ_1) and (μ_λ, σ_2) are *isomorphic* if the GEMs they represent are combinatorially equivalent; that is, their nodes, edges and colourings are equal after reshuffling.

Definition 3.5 (Symmetry)**.** Two pairs of permutations (μ_λ, σ_1) and (μ_λ, σ_2) are *symmetric* if they are isomorphic AND if $\sigma_1 = \sigma_2$ when their GEMs are rearranged into extended standard form (through a combination of node swaps and colour swaps).

Informally, isomorphisms can be thought of as "disguises" for a GEM, symmetries as copies of each unique disguise, and our desired weight function $1/w$ as counting the number of unique disguises (and then reciprocating it). We now use results of group theory to do this counting.

For each $\tau \in S_3/\{\alpha \in S_3 : \alpha \text{ is a colour swap symmetry}\}$, it turns out that $\prod_{j=1}^{n} j^{a_j} \cdot a_j!$ is the number of ways that μ_λ can be reshuffled, where the j's and a_j's form the cycle structure given by the integer partition λ. In group theory jargon, this number is exactly the cardinality of the centraliser c_{μ_λ}.

We also denote $|\mathrm{Sym}(\mu_\lambda, \sigma)|$ to be the number of symmetries associated with (μ_λ, σ). Then, our weight function is

$$\frac{1}{w(\mu_\lambda, \sigma)} = \frac{2}{|\mathrm{Sym}(\mu_\lambda, \sigma)|} \sum_{\tau} |c_{\mu_\lambda}|.$$

The multiplication by 2 accounts for reflection symmetries. Thus, given (μ_λ, σ), we add its corresponding triangulation to the sample with weight $1/w(\mu_\lambda, \sigma)$.

3.4 Python implementation

We used the Python programming language (Python Core Team, 2020) along with functions from external libraries including `sympy`, `numpy` to implement our sampling procedure. We included functions to

- uniformly sample integer partitions
- convert partitions to permutations
- uniformly sample permutations from S_n,

and created a `Triangulation` class that took two permutations as inputs, with useful attributes and callable methods that computed things such as the

- GEM and storing it as a dictionary data structure
- genus of the surface represented by the triangulation (see Definition 4.1)
- number of symmetries
- weight
- isomorphism signature.

The code containing the above class and functions can be found at the following GitHub repository: https://github.com/rajan-shankar/Uniform-Sampling-Procedure-for-Triangulations.

4 Experimental results

In this section, we run our sampling procedure over different values of n and collect data of interest for each sampled pair of permutations. We then visualise the data and form some simple conjectures.

4.1 Data collected

We used the `pandas` and `matplotlib` libraries to store and plot our data, respectively. The main data point of interest for each pair of permutations was the *genus* of the surface it represented.

Definition 4.1 (Genus)**.** The *genus* of a surface tells us how many "holes" the surface has. That is, a sphere has 0 holes so it is of genus 0, a torus has 1 hole so it is of genus 1, a double torus has 2 holes so it is of genus 2, and so on.

Theorem 4.1 (Classification of surfaces)**.** It is a well known fact of topology that every closed and orientable surface is homeomorphic to one of the aforementioned surfaces.

The genus g of a surface represented by a triangulation can be calculated via the Euler characteristic $\chi = v - e + f$, where v, e and f are the number of vertices, edges and faces in the triangulation, respectively. Since it is easy to see that $f = 2n$ and $e = 2n/2 = n$, the Euler characteristic boils down to $\chi = v - n$. Then, the genus is calculated as $g = \frac{2-\chi}{2} = \frac{2-(v-n)}{2}$, which really only depends on v. v can be calculated based on the number of cycles in each bi-coloured permutation, as summarised in the following proposition.

Proposition 4.1 (Vertices and total number of cycles)**.** The number of vertices in a triangulation is equal to the total number of cycles among the three permutations μ_λ, σ and $\phi = \mu_\lambda^{-1} \circ \sigma$.

Proof. By construction of a GEM, there is a one-to-one correspondence between a bi-coloured cycle and a vertex. Thus, counting up the number of bi-coloured cycles results in the number of vertices in the triangulation. Since each cycle in each of the three permutations represents a bi-coloured cycle, we have that the total number of cycles among these permutations is equal to the number of vertices in the triangulation. □

We also collected data on the number of symmetries, and whether the sampled pair of permutations represented a disconnected GEM.

4.2 Computation time

The sampling procedure script was run on a powerful, research-purpose computer at the School of Mathematics and Statistics at The University of Sydney. We collected 1,000,000 samples at each value of $n = 3, 4, 5, \ldots, 139, 140$. For each value of n, we recorded the time it took to do one million samples and plotted it in Figure 4.1.

The jumps at $n = 31$ and $n = 81$ are simply artefacts of the sampling procedure being conducted in three blocks. What's more important is that the time taken to obtain one million samples appears to increase linearly with n. Thus, we could repeat the process for larger n and expect the running time to not increase drastically.

4.3 Genus distributions

For every value of n, we computed the empirical genus distribution. Figure 4.2 displays these distributions across the selection of values $n = 3, 10, 20, 50, 90, 140$.

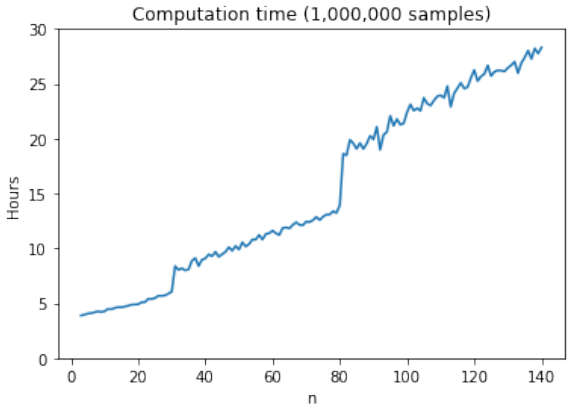


Figure 4.1: Time required in hours to sample one million triangulations for each value of $n = 3, \ldots, 140$. Appears to increase linearly with n. The jumps are simply artefacts of the sampling procedure being conducted in three blocks.

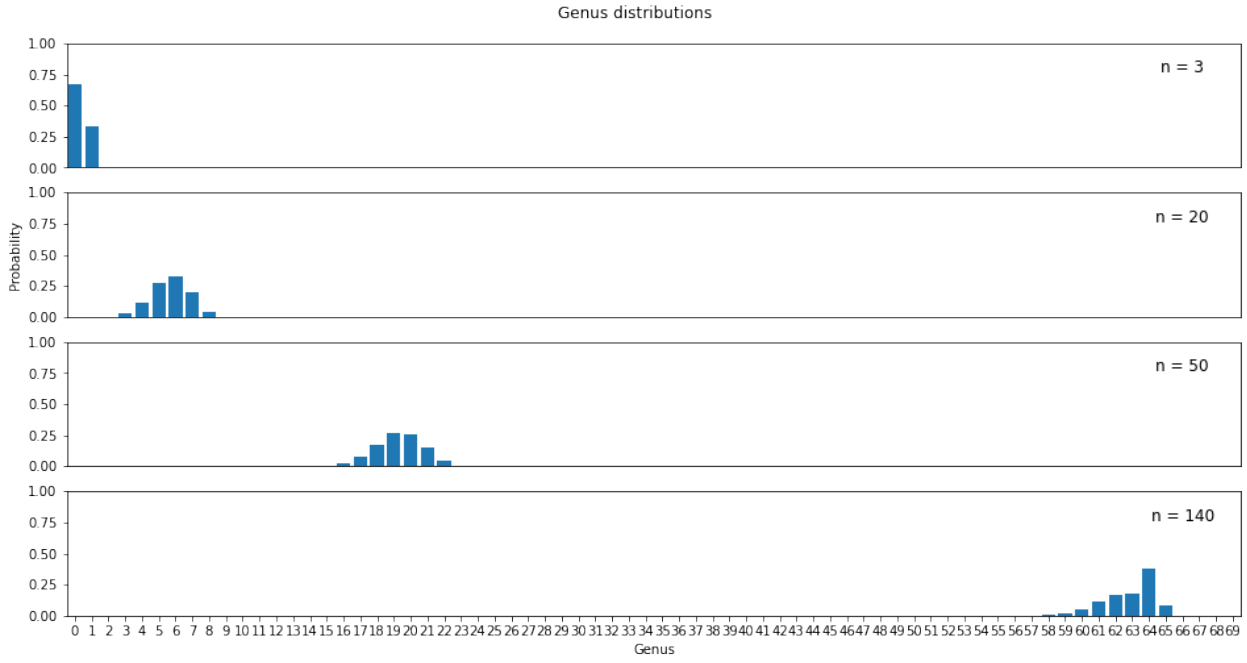


Figure 4.2: Empirical distributions of genera for four different values of n. Most of the probability mass lies on $\sim$5 genera.

The maximum possible genus for a given n is $(n-2)/2$ if n is even, and $(n-1)/2$ if n is odd. It is interesting to note that as n increases, the bulk of the probability mass is located on relatively few genera, slightly away from the maximum genus.

We can investigate this further by looking at how much probability mass is within the interval $[\bar{x}-2, \bar{x}+2]$, where $\bar{x}$ is the mean (expected value) of a genus distribution, and plotting this number against n as shown in

Plot A of Figure 4.3.

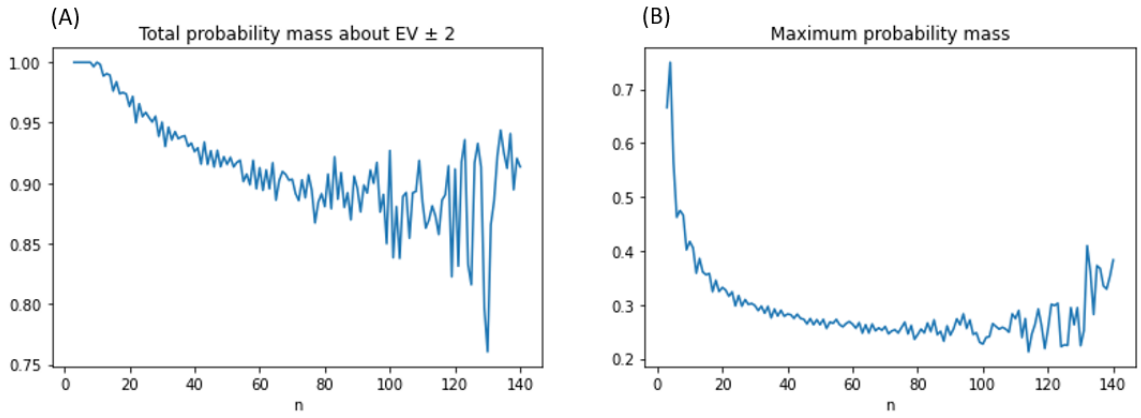


Figure 4.3: Plots used to investigate the trend in the shape of the genus distributions of Figure 4.2 as n increases. Plot A graphs the total probability mass located within two units of the distribution mean for each n. Plot B graphs the maximum probability mass located on a single genus for each n.

The probability mass within a fixed interval of length 4 decreases until around $n = 80$, stabilising at a total probability mass of around 0.9. The amount of fluctuation, or noise, also increases with n, and this can be attributed to our sample size of one million not being representative (large enough) as n becomes large.

Alternatively, instead of the sum of the probability mass within a fixed interval, we may plot the maximum probability mass for each value of n. This is shown in Plot B of Figure 4.3.

We observe a similar stabilising pattern after $n = 80$ where the maximum probability mass remains around 0.25, along with an increase in noise.

4.4 Average number of symmetries

Another piece of data we collected was the average number of GEM symmetries for each value of n. Figure 4.4 reveals that this number rapidly decays as n increases.

We may suspect that the decay is exponential. A property of the exponential curve is that it is *memoryless*, meaning, informally, that the curve looks visually the same when zoomed in upon. In Figure 4.5, we plot the average number of symmetries for $n = 3, \ldots, 12$ (Plot A) and for $n = 13, \ldots, 22$ (Plot B).

It is quite clear that the average number of symmetries as a function of n portrays this memoryless behaviour, and thus we have reason to suspect that this decay is exponential-like.

What's interesting is that for $n \geq 26$, every single sampled pair of permutations corresponded to a GEM with only one symmetry. This highlights how rare it is to find symmetric GEMs for large n.

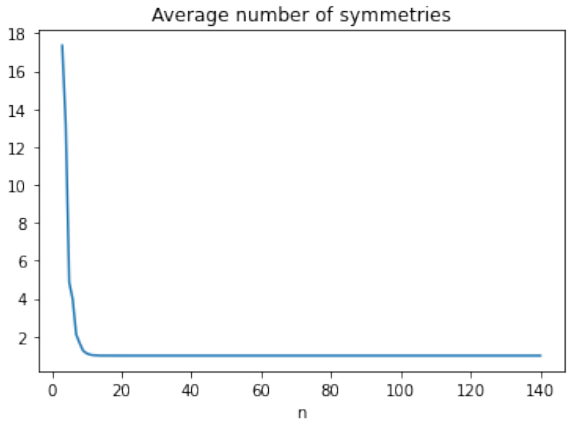


Figure 4.4: Average number of symmetries for each n.

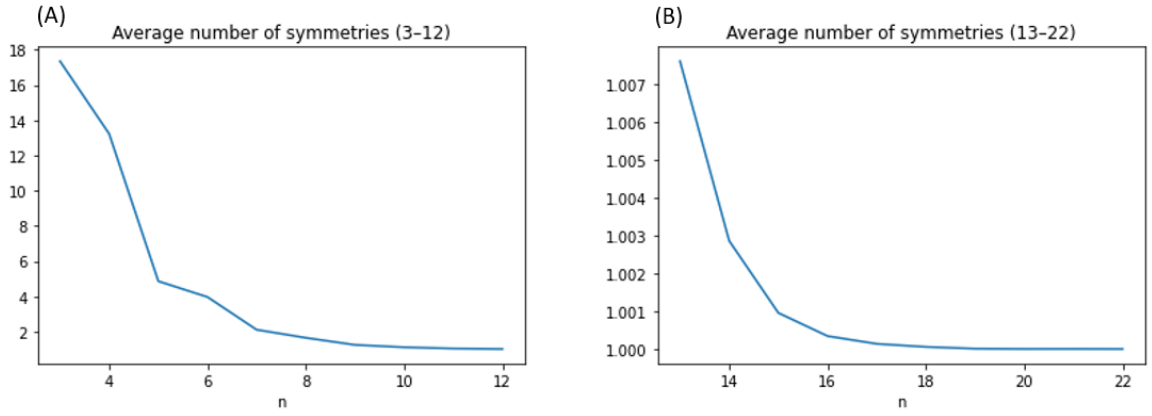


Figure 4.5: Plots used to investigate whether the decay in the average number of symmetries is exponential-like. Plot A graphs this for $n = 3, \ldots, 12$ and Plot B graphs this for $n = 13, \ldots, 22$. The plots look visually similar, i.e. they exhibit the memoryless property that characterises exponential decay.

4.5 Proportion of GEMs that are disconnected

Here, we wish to get a feeling for the probability that the subgroup generated by a sampled pair of permutations (μ_λ, σ) is transitive. There exist literature that provide a lower bound for two permutations uniformly sampled from S_n. Dixon (1969) proved that the probability that two such permutations generate the *entire* symmetric group S_n tends to $3/4$ as $n \to \infty$. This is a lower bound because there exist other subgroups that are transitive apart from just S_n.

However, we must remember that (μ_λ, σ) is NOT being sampled from the space $S_n \times S_n$ since μ_λ is sampled from the set of cycle structures. Thus, Figure 4.6 provides us with novel insight into the transitive-subgroup-generation probability for when μ_λ is sampled from the set of cycle structures and σ from S_n.

We observe that the proportion of subgroups that are intransitive appears to converge to a (relatively small)

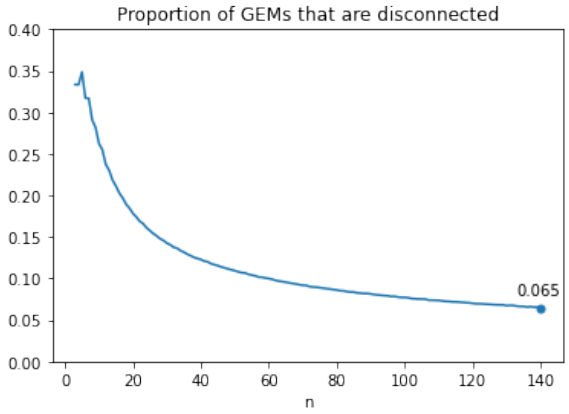


Figure 4.6: Proportion of GEMs that are disconnected for each n. That is, the proportion of sampled permutation pairs that did not generate a transitive subgroup for each n.

constant. This is a useful result in the run-time analyses of algorithms that rely on sampling GEMs—we can expect to sample a GEM that is connected with high probability.

5 Conclusion

In this report, we worked towards a uniform sampling procedure for abstract triangulations of surfaces. We first developed the mathematical background by noting key definitions and concepts. We then proposed a uniform sampling procedure where we dealt with the lack of a one-to-one correspondence between pairs of permutations and triangulations by detecting the occurrence of disconnected GEMs, and by introducing weights as a function of isomorphisms and symmetries. We also discussed reducing the sample space of μ to that of μ_λ (canonical permutations of cycle structures) to improve the performance of the sampling procedure. Finally, we visualised and discussed some experimental results based on a Python implementation of the uniform sampling procedure.

One experimental result concerned the distribution of the genera of surfaces represented by the sampled triangulations. As n increased, the probability mass within 2 units of the mean of the genus distribution stabilised at around 0.9. We also observed that the maximum probability mass on a single genus stabilised at around 0.25. Another finding was that the average number of symmetries decayed exponentially as n increased. Finally, we observed that the proportion of GEMs that were disconnected decreased to a relatively small number (< 0.065).

Future work could involve conducting statistical hypothesis tests or constructing mathematical proofs for some of the trends revealed in our experimental results. Alternatively, different data could be collected from the samples, such as *vertex degree sequences*. The sampling procedure may also be extended to triangulations of higher-dimensional manifolds than just surfaces.

Acknowledgements. This research was funded by the Australian Mathematical Sciences Institute as part of the Vacation Research Scholarships 2021-22 program. Guidance and advice was provided by Dr Jonathan Spreer from The University of Sydney. The powerful research computer used to collect millions of triangulation samples was provided by the School of Mathematics and Statistics at The University of Sydney.

References

Dixon, J. D. (1969). The probability of generating the symmetric group. *Mathematische Zeitschrift*, *110*(3), 199–205.

Andrews, G. E. (1977). *Theory of partitions*. Longman Higher Education.

Cavicchioli, A., Grasselli, L., & Pezzana, M. (1980). A normal decomposition for closed n-manifolds. *Boll. Un. Mat. Ital. B (5)*, (3), 1146–1165.

Fristedt, B. (1993). The structure of random partitions of large integers. *Transactions of the American Mathematical Society*, *337*(2), 703–735.

Spielman, D. A., & Teng, S.-H. (2009). Smoothed analysis. *Communications of the ACM*, *52*(10), 76–84.

Artin, M. (2010). *Algebra* (2nd ed.). Pearson.

Python Core Team. (2020). *Python: A dynamic, open source programming language*. Python Software Foundation. https://www.python.org/