Navigating the Pachner Graph: Algorithms for Searching and Sampling Triangulations

Daniel Bruwel

An essay submitted in partial fulfillment of
the requirements for the degree of
B.Sc. (Honours)

Pure Mathematics
University of Sydney

October 2025

CONTENTS

Introduction

The process of decomposing a space or surface into discrete units such as triangles and tetrahedra is a standard technique used in a range of disciplines to take the analytic problem of studying a space or surface, to a combinatorial problem that opens it up to a range of computational techniques. Some examples of where triangulations are used includes physics such as the Regge calculus of general relativity [**?**] and the causal dynamic triangulations of quantum gravity [**?**] where spacetime is triangulated for simulations; engineering where fluid and solid dynamics use "meshes" (triangulation) to approximate wings or bridges so that each component can be studied easily [**?**]; computer graphics where objects are rendered with triangulations [**?**]; and data science where the data can be used to generate a triangulation that can then be explored to reveal more about the system [**?**]. Triangulations also have many important purposes within maths itself, where they can be used to compute topological properties like homology groups and the Euler characteristic [**?**]; differential geometry with fields such as combinatorial Ricci flows [**?**]; and graph theory where concepts from triangulations where used in proofs such as the proof of the four colouring theorem [**?**]. The main challenge with triangulations of surfaces and spaces, is that they are not unique, and different triangulations of a surface are better for different tasks. For example, finding the triangulation with the least faces helps in algebraic topology [**?**], whereas in computer graphics the triangulation that best approximates the geometry of the original surface is preferred. As such, considerable effort has been put into trying to explore and understand the world of triangulations of a given surface and find techniques to generate "good" triangulations.

While there are a range of different notions of what is meant by good, this report primarily explores topological and combinatorial properties of triangulations. The techniques can undoubtedly be extended to a range of different notions of goodness, but topological and combinatorial properties are fundamental to the triangulation itself, and do not depend on the specific geometry like the angles within or the size of specific triangles in the triangulation. This has the advantage of narrowing the focus of exploration to abstract triangulations which are discrete and countable, improving computational manageability and ensuring that the results are general and only dependent on the triangulation itself, not other erroneous geometric factors. As such, this report primarily focuses around exploring properties of the triangulations of a single vertex $3-$sphere that arise from the connection between such a triangulation, and a set of knots related to the edges within the triangulation. Looking for triangulations that relate to interesting and complex knots.

The primary objective of this report is to implement, analyse, and compare a range of different computational techniques for searching the space of triangulations. We investigate classical search and sampling techniques, namely Markov Chain Monte Carlo, greedy search, and simulated annealing. While greedy search algorithms have been a central tool to the study of this problem, Markov Chain Monte Carlo is a relatively new technique [**?**] and we introduce the application of simulated annealing to this problem for the first time. Additionally, we introduce the groundwork for a new approach that uses transformer-based models to explore the space, opening the door for reinforcement learning which is steadily developing a repertoire of success in a range of different problems, including learning to play games like Go [**?**], finding improved sorting algorithms [**?**], finding improved solutions to 4×4 matrix multiplication and kissing number problems [**?**], and helping with the development of new conjectures [**?**].

The report is structured as follows. Chapter 1 brings together the necessary preliminaries, including manifolds, some knot theory, and an overview of various statistical and machine learning techniques. Chapter 2 provides a detailed outline and discussion of the specific algorithms that we are implementing, including justifications for the choices we made. Chapter 3 presents a description and analysis of a series of numeric experiments run across 5 different search problems and compares them. Finally, chapter 4 concludes with a discussion of our findings and outlines the direction for future research.

CHAPTER 1

Preliminaries

1.1. Mathematical Preliminaries

1.1.1. Manifolds. A $n-$*dimensional manifold* or a $n-$*manifold* is a topological generalisation of a surface, they are topological spaces that are "well behaved" and locally homeomorphic to $\mathbb{R}^n$. We recall the following basic definitions before precisely defining a manifold.

Definition 1.1 (Second Countable)**.** *A Topological space $(\mathcal{T}, \tau_{\mathcal{T}})$ is **Second Countable** if the topology $\tau_{\mathcal{T}}$ of $\mathcal{T}$ has a countable base.*

Definition 1.2 (Hausdorff)**.** *A Topological space $(\mathcal{T}, \tau_{\mathcal{T}})$ is **Hausdorff** if for any two points $x \neq y$ in $\mathcal{T}$ there are neighbourhoods U of x and V of y such that $U \cap V = \varnothing$*

With these we can define

Definition 1.3 ($n-$dimensional manifold)**.** *A $n-$**dimensional manifold** is a Hausdorff, second countable, topological space $\mathcal{M}$ where for every point $p \in \mathcal{M}$ there is a neighbourhood $U(p)$ of p that is Homeomorphic to an open subset of Euclidean space $\mathbb{R}^n$*

Remark 1.4. We have used a Homeomorphism to an open set of R^n, this is equivalent to saying that each point is either isolated (if $n = 0$) or has a neighbourhood that is Homeomorphic to all of $\mathbb{R}^n$

Remark 1.5. We will refer to these simply as Manifolds, compared to other areas of research these may be called *topological manifolds* to emphasise that they do not have any further structure (c.f. *differentiable manifolds, Riemannian manifolds, etc.*)

Some examples of Manifolds include the circles S^1, the sphere S^2, the Torus T^2, Euclidean space $\mathbb{R}^n$. Some non-examples include the disk D^2 because its boundary has no neighbourhood that is homeomorphic to an open subset of $\mathbb{R}^n$, the figure-eight curve because at the "crossing" there is no way to deform it to look like an open subset of $\mathbb{R}$, or the line with two origins because this is clearly not Hausdorff.

In the study of triangulations we are often interested in a specific additional structure that a manifold can admit known as a *piecewise-linear structure* or *PL-structure*. Intuitively, a PL-structure allows a manifold to be realised as a collection of "flat pieces" joined together in in a linear fashion. For example S^1 admits PL-structure that can be realised as the boundary of a triangle, or a square, etc. We say that two PL-manifolds (manifolds with PL-structure) are *PL-homeomorphic* if

they can be transformed into each other via a homeomorphism that is itself piecewise linear. This is all formally defined in terms of charts, atlases, and transition maps which can be found in most introductory books on geometric topology (e.g. "Introduction to piecewise-linear topology " [**?**]).

A particularly useful result due to Radó [**?**] and Moise [**?**] says that for dimension 3 or less PL-manifolds and PL-homeomorphisms are the "same" as manifolds and homeomorphisms. More precisely

Theorem 1.6 (Hauptvermutung)**.** *For dimension 3 or less, every manifold admits a unique PL-manifold up to PL-homeomorphism.*

Remark 1.7. This theorem does not hold for *four* or more dimensions [**?**].

We will be considering manifolds of dimension 3, so will often talk of a manifold and the unique class of PL-structures (up to PL-homeomorphism) that the manifold can admit as being the same thing.

1.1.2. Triangulations. We begin by defining *simplices* - the fundamental building blocks of triangulations

Definition 1.8 (simplices)**.** *An $n-$simplex Δ is the $n-$dimensional convex hull of $n+1$ vertices.*

A $0-$simplex is a point, a $1-$simplex a line, a $2-$simplex a triangle, and a $3-$simplex a tetrahedron. One way to construct an $n-$simplex is by taking the *join* of n points where we recall that

Definition 1.9 (Join)**.** *For two topological spaces X and Y, the join denoted $X * Y$ is constructed by taking $X \times Y \times [0,1]$ and quotienting by the equivalence relation that $(x,y,0) \sim (x,y',0)$ and $(x,y,1) \sim (x',y,1)$ for $x, x' \in X$ and $y, y' \in Y$.*

Any subset of $i+1$ points of an $n-$simplex forms another simplex that we call an $i-$*dimensional face*.

We can takes simplices and "glue" them together to form more complex geometric structures. Formally this is done via *face gluings*

Definition 1.10 (face gluing)**.** *For two $n-$simplices Δ_1 and Δ_2 and two $i-$dimensional faces $F_1 \subseteq \Delta_1$ and $F_2 \subseteq \Delta_2$ we define the **face gluing** of F_1 to F_2 by taking $\Delta_1 \sqcup \Delta_2 / \sim$ where $\sim$ is defined by a gluing map which is a homeomorphism between F_1 and F_2*

We typically create the gluing map by taking a bijection between the vertices of F_1 and F_2 and linearly interpolating to create the full homeomorphism.

Jonathan - I'm not so sure about the following - is this as general as what we were dealing with? Does it allow for an edge to form a loop, or multiple edges to go between the same vertices etc?

Definition 1.11 (PL-Sphere)**.** *The PL $0-$sphere is a pair of isolated points. For integers $n > 0$ the PL $n-$sphere is a PL manifold homeomorphic to the $n-$sphere.*

Definition 1.12 (PL-Triangulation)**.** *Construct $\mathcal{T}$ from a finite collection of $n-$simplices $\{\Delta_1, ..., \Delta_k\}$, called facets, by gluing their $n-1-$dimensional faces together. We call this a PL-triangulation if the following conditions are met*

a) If a face is glued to itself, the face gluing must happen along the identity map.
*b) For all vertices v, the **link** defined as the set of all simplices Δ such that the join $v * \Delta$ is in $\mathcal{T}$ forms a PL $n-1-$sphere*

This is a PL-triangulation of a manifold $\mathcal{M}$ if there is a homeomorphism from $\mathcal{T} \to \mathcal{M}$.

Remark 1.13. We can view PL-manifolds (manifolds with PL-structure) and PL-triangulations as the same thing in some sense. That being that for every PL-manifold there is a PL-triangulation that compatible with it, and every PL-triangulation is a PL-manifold.

1.2. Knot Theory

Definition 1.14 (Ambient Isotopy)**.** *For a pair of manifold N, M and embeddings g, h of N into M, an ambient isotopy from g to h is a continuous map $F : M \times [0, 1] \to M$ such that $F_t : M \to M$ is a homeomorphism, F_0 is the identity, and $F_1 \circ g = h$*

Definition 1.15 (Knot)**.** *A knot is a smooth embedding of S^1 into S^3*

Remark 1.16. We somewhat loosely refer to a knot as both the actual embedding function, and its image in S^3. However, if we say "a knot K", we typically mean the image of the embedding.

Definition 1.17 (Knot Equivalence)**.** *Two knots are said to be equivalent if there is an ambient isotopy between their embedding functions.*

Definition 1.18 (Knot Complement)**.** *For a knot K, the tubular neighbourhood $\nu(K)$ is a small, closed, 3-dimensional region surrounding K that is homeomorphic to the solid torus $S^1 \times D^2$. The knot complement is $X_K = S^3 \backslash \nu(K)$*

We have that the boundary $\partial x_K \cong T^2$ is a torus.

Definition 1.19 (Infinite Cyclic Cover)**.** *For the knot complement X_K, $\tilde{X}_K$ a cover of X_K is an infinite cyclic cover if $\mathrm{Deck}(\tilde{X}_K / X_K) \cong \mathbb{Z}$*

Every knot has a unique infinite cyclic cover. This follows from the fact that $H_1(X_K; \mathbb{Z}) \cong Z$ and the properties of covering spaces.

Definition 1.20 (Alexander Module)**.** *For a knot K, take the infinite cyclic cover $\tilde{X}_K$ of the knot complement. The deck transformation group of this space is isomorphic to $\mathbb{Z}$, generated by some transformation t. Form the ring of Laurent polynomials $\Lambda = \mathbb{Z}[t, t^{-1}]$. The Alexander module is the first homology group $H_1(\tilde{X}_K)$ viewed as a module over Λ.*

Definition 1.21 (Elementary Ideal)**.** *For a module M over a unique factorisation domain R with presentation of n generators g_i and m relations $r_j = a_{j1}g_1 + a_{j2}g_2 + \cdots = 0$ we construct an $m \times n$ matrix A of all the relations. The kth elementary ideal is the ideal $E_k(M)$ generated by all the $(n-k) \times (n-k)$ minors of A.*

Theorem 1.22. *The elementary ideals are independent of how the module is presented. So we refer to the elementary ideals of the module without reference to the presentation.*

Definition 1.23 (Alexander Polynomial)**.** *The Alexander Polynomial is the generator of the first elementary ideal of the Alexander module.*

Theorem 1.24. *The Alexander polynomial is a knot invariant (the same for equivalent knots).*

The proof of the above is due to J. W. Alexander [**?**].

1.2.1. Isomorphism Signatures. We say that two PL-triangulations are *combinatorially isomorphic* if they are the same up to relabelling of vertices. Because there is no interesting mathematical information in the labelling, we typically define a canonical labelling. While the exact details of constructing a canonical labelling are specific and can depend on the software choice, the general idea is as follows.

a) For each facet, calculate some label independent invariant (e.g. the number of unique facets being glued to).
b) Partition the facets into labelled bins based on this invariant (this labelling is canonical based on the value of the invariant)
c) For each facet in each bin (containing multiple facets), look at which bins the facets neighbouring (from gluing) facets are in, use this information to construct a new label, if there are facets in the same bin that get a different new label, split the bin in some canonical way and create a new labelling.
d) Repeat the above process, constructing a splitting tree. This tree is canonical. This tree is not guaranteed to split the bins into singletons. If it gets "stuck", make an arbitrary choice and continue the algorithm until everything falls into a singleton. This is a possible canonical labelling.
e) If an arbitrary choice had to be made (or perhaps multiple), rerun the entire algorithm for each possible choice that could have been made, creating a small list of possible canonical labelling. Chose the lexicographically smallest labelling.

The choice of canonical labelling used for the rest of this paper is as is in regina [**?**]. The details of the initial invariant, how the bins are labelled, and optimisations can be found in the corresponding paper [**?**]. In the worst case scenario this process takes $\mathcal{O}(n! \cdot k^2)$ where n is the dimension and k is the number of facets. For a $3-$dimensional triangulation the $n!$ term is small and this algorithm is fast.

Once a canonical labelling is found, an "isomorphism signature" is calculated. For a $3-$dimensional triangulation, the only information that needs to be stored is the number of tetrahedra, and for face of each tetrahedron what the destination tetrahedron is, and what the vertex order is. We do not need to specify the destination face because this can be determined by a combination of permutation labelling, looking at where the destinations faces are being mapped, and strict ordering. To efficiently store this, the following is done.

a) For each face of each tetrahedron, calculate $v_i = 6 \cdot$ destination id $+$ permutation id. This is valid due to their being 6 vertex permutations. This value will less than $6k$ for k facets.
b) Create a sequence $(v_0, v_1, \cdots, v_{4k-1})$ ordered based on the canonical labelling.
c) Compute $I = v_0 + v_1(6k) + v_2(6k)^2 + \cdots$
d) Convert I to base 64 and store as a string.
e) Convert k to some string and append to the front of the string representation of I.

This process form a string / number representing the unique combinatorial isomorphism signature for any given triangulation of a $3-$manifold (practically there are size restrictions on the number of facets k).

1.2.2. Pachner Moves and the Pachner Graph. A Pachner move is a specific process that changes a PL-triangulation without changing the underlying topological space.

Definition 1.25 (Complementary Triangulation)**.** *Let Δ a $d+1$simplex, let A be a a connected sub complex of $\partial\Delta$ - that is a set of $d-$simplexes that form a connected topological space. The complementary triangulation of A is $B = \partial\Delta \backslash A$.*

From this we can abstractly define a Pachner move

Definition 1.26 (Pachner Move)**.** *For a d-dimensional PL-Manifold, identify an $i-$dimensional simplex σ^i, the set of all facets that contain σ^i form a sub complex of the boundary of some $d+1$dimensional triangulation. Replace this set with the complementary triangulation of this sub complex. This Pachner move is called an $i-$move.*

Theorem 1.27. *Two triangulations $\mathcal{T}$ and $\mathcal{T}'$ are reachable in finitely many Pachner moves if and only if they represent the same PL-manifold.*

For triangulations of $3-$manifolds, we can take $i = 0, 1, 2, 3$, where $i = 0, 3$ are inverses of each other, and likewise $i = 1, 2$. For a $2-$move, we identify a triangle and find all the tetrahedra that share that triangle, there are two of these, we replace these with 3 tetrahedra that now share a common edge. The $1-$move is the inverse of this. We often call these moves $(3, 2)$ and $(2, 3)$ moves to explicitly state how we are going, for example, from 3 tetrahedra to 2. For a $0-$move, we identify a vertex with 4 tetrahedra that share it, you replace these with a single tetrahedron. The $3-$move is the inverse. These are often called $(4, 1)$ and $(1, 4)$ moves as above.

Because all triangulations of a specific PL-manifold can be reached through Pachner moves, we define

Definition 1.28 (Pachner Graph)**.** *For a given PL-manifold $\mathcal{M}$, the Pachner graph for $\mathcal{M}$ us the graph (V, E) given by $V = \{$triangulations of $\mathcal{M}\}$ and for $v_i, v_j \in V$, $e_{ij} = (v_i, v_j) \in E \iff v_i, v_j$ are related via a single Pachner move.*

Because of the above theorem, the Pachner graph is fully connected for any PL-manifold. The Pachner graph is however, infinite. We additionally endow the Pachner graph with "levels" where each level encodes the number of facets in the triangulation.

Theorem 1.29. *The Pachner graph grows super exponentially in its level.*

We notice that $(4, 1)$ and $(1, 4)$ moves change the number of vertices, but the $(3, 2)$ and $(2, 3)$ moves do not. We are often interested in "single vertex" triangulations, for these we can exclusively use $(3, 2)$ and $(2, 3)$ moves. A particularly useful result is that

Theorem 1.30. *Any single vertex triangulation of the $3-$sphere S^3 can be reached through only $(3, 2)$ and $(2, 3)$ moves.*

We analogously define the Pachner graph for $1-$vertex triangulations of S^3, which is once again a fully connected graph. Unlike the general Pachner graph, it is currently unknown if the Pachner graph for $1-$vertex triangulations of S^3 grow super exponentially or exponentially in its level.

1.3. Machine Learning Preliminaries

1.3.1. Computational Complexity.

Definition 1.31 (Time Complexity)**.** *For an algorithm with input size n, the time complexity is the asymptotic worst case runtime of the algorithm, written in "Big-O" notation as $\mathcal{O}(g(n))$.*

Definition 1.32 (Space Complexity)**.** *For an algorithm with input size n, the space complexity is the asymptotic worst case amount of memory the algorithm needs to allocate, written in "Big-O" notation as $\mathcal{O}(g(n))$.*

Definition 1.33 (Decision Problem)**.** *A decision problem is a type of problem that can be answer as either "yes" or "no"*

Definition 1.34 (Decidable)**.** *A decision problem is **decidable** if there exists an algorithm that can always provide the correct answer for any given input in a finite amount of time. Conversely the problem is **undecidable** if it is not decidable.*

Remark 1.35. While most problems are not decision problems, it is often possible to convert many problems into a decision problem. For example, if the problem is to multiply two numbers together, we can convert this to a decision problem of "is the ith binary digit a 0?", this can be repeated for all digits.

Definition 1.36 (Complexity Class)**.** *A complexity class is a class of problems that can be solved under some model of computation within some resource bound i.e. some time or space bound. A problem belongs to a given complexity class if there exist an algorithm to solve the problem within this bound. For "turning machines", the main complexity classes are*

a) L*: The problem can be solved using a logarithmic amount of space.*
b) NL*: A solution can be verified using a logarithmic amount of space.*
c) P*: The problem can be solved in polynomial time.*
d) NP*: A 'yes' solution can be verified in polynomial time.*
e) $\mathrm{co}\,NP$*: A 'no' solution can be verfied in polynomial time.*
f) $PSPACE$*: A solution can be found using a polynomial amount of space.*
g) $EXPTIME$*: A solution can be found using an exponential amount of time.*

There exist many other complexity classes for both turning machines and other models for computation.

Remark 1.37. NP and $\mathrm{co}\,NP$ are distinct. For example for a graph G, the problem of if there is a path that visits all nodes exactly once is NP because I can confirm that a path exists if you simply give me one, but it is believed to not be $\mathrm{co}\,NP$ as this would mean I could easily verify based on some example path that there can't be a path that visits all nodes exactly once.

Remark 1.38. A turning machine is a common model for computation, we do not define it here but there are many resources available.

Remark 1.39. We have defined NL and NP as having solutions verifiable in a specific bound, this means that if the question is for example, is there a path of length k between two nodes, someone can give a path that is a solution, and it is possible to verify that this connects the two nodes and has length k. Formally NL and NP are not defined this way, but instead using things called "non-deterministic Turing machines", but the definitions can be shown to be equivalent.

Theorem 1.40. $L \subseteq NL \subseteq P \subseteq NP \subseteq PSCPAE \subseteq EXPTIME$

Remark 1.41. There are conjectures that some of these subsets may be proper subsets or equalities, most famously the question of if $P = NP$, but there are not as of yet any solutions.

Definition 1.42 (Hard Problem)**.** *For a given complexity class* C*, a problem* P *is called a* $C-$*hard problem if every problem* $P' \in C$ *can be "reduced" to* P *via some algorithm that is "efficient" relative to* C*.*

Remark 1.43. We do not formally define what it means to be "efficient" relative to C as it is getting into the details too much, but for NP and $PSCPAE$ problems, a polynomial time reduction is efficient.

Definition 1.44 (Complete Problem)**.** *For a given complexity class C, a problem P is called a $C-$complete problem if it is a $C-$hard problem and $P \in C$*

Practically, problems that belong to P can be solved tractable on a modern computer, we can typically scale resources and solve the problem. However, all the algorithms we have for NP complete problems require exponential resources to run, which becomes computationally infeasible quickly. As such, if we find a problem to be $NP-$complete or harder, it means we can't just "throw more compute" at the problem to solve it and may have to resort to suboptimal solutions using heuristics etc.

From these definitions we can discuss a few important problems that we will encounter, and their respective complexity. The general high level of complexity of many of these problems is a motivating factor for our algorithms.

Theorem 1.45. *Finding the shortest path between two vertices v_i and v_j of the Pachner graph is PSPACE complete.*

Theorem 1.46. ?? *Deciding if a presentation of a group is trivial is undecidable.*

Theorem 1.47. *Deciding if $3D$ triangulation is S^3 is in $NP \cap \text{co}\, NP$*

Theorem 1.48. *Checking if a knot is the unknot is $NP \cap \text{co}\, NP$*

1.3.2. Markov Chain Monte Carlo.

Definition 1.49 (Markov Chain)**.** *A Markov Chain is a sequence of values $x(t)$ indexed by "time" where the sequence has the Markov Property - that is $x(t+1)$ depends only on $x(t)$, typically probabilistically.*

Monte Carlo methods are a class of methods that use random sampling to approximate something. Often, they are used to approximate the integral of some complex function, however not exclusively.

Markov Chain Monte Carlo is a class of methods to sample from a distribution by constructing a Markov Chain that's steady state solution is the distribution. More formally we define

Definition 1.50 (Markov Transition Kernel)**.** *Take $(\mathcal{X}, \mathcal{F})$ a measurable space with $\mathcal{F}$ a sigma algebra. A Markov Transition Kernel is a function $P : \mathcal{X} \times \mathcal{F} \to [0, 1]$ such that*

a) for any $x \in \mathcal{X}$, the function $A \mapsto P(x, A)$ is a probability measure on $(\mathcal{X}, \mathcal{F})$

b) for any $A \in \mathcal{F}$, the function $x \mapsto P(x, A)$ is a measurable function.

The Markov Transition Kernel is used to "evolve" the state. That is, if you are currently "located" at $x \in \mathcal{X}$, $P(x, A)$ tells you the probability of "ending up" in A. More precisely, if at time t the system is distributed according to ν_t, then the distribution at $t + 1$ is

$$\nu_{t+1} = \nu_t P(A) = \int_{\mathcal{X}} P(x, A) d\nu_t(x) \tag{1.51}$$

Definition 1.52 (Target Distribution)**.** *π is called a target distribution for a Markov Transition Kernel P if the following two conditions hold*

a) ***Stationarity****: $\pi P = \pi$*
b) ***Ergodicity****: For $\pi-$almost all points x_0,*

$$\lim_{n\to\infty} \|P^n(x_0, \cdot) - \pi(\cdot)\|_{TV} = 0$$

Remark 1.53. A statement is true for "$\pi-$almost all x_0" means that it is true for all x_0 except a set that has measure 0 under π

Typically proving stationarity and ergodicity is difficult, so the following is used instead

Theorem 1.54. *For a Markov transition kernel P, if the following are true for some distribution π*

a) ***$\pi-$irreducibility****: For any $A \in \mathcal{F}$ with $\pi(A) > 0$, there exists some n such that $P^n(x, A) > 0$ for all $x \in \mathcal{X}$*
b) ***Aperiodicity****: The chain is aperiodic*
c) ***Recurrent****: For any measurable set $A \in \mathcal{F}$, and any starting point $x \in \mathcal{X}$ the hitting time τ_A is finite almost surely, where τ is the amount of steps to go from x to A.*
d) ***Reversibility****: For any two $A, B \in \mathcal{F}$ we have*

$$\int_A P(x, B)d\pi(x) = \int_B P(x, A)d\pi(x)$$

then π is the target distribution for the Markov transition kernel P.

In particular, if we have some process $\hat{P}$ that takes in some $x(t)$ and produces some $x(t + 1) = \hat{P}(x(t))$ according to some Markov transition kernel P, we can create a Markov Chain, by continuously generating samples, and applying $\hat{P}$ to them recurrently, these samples will be "distributed according to" π. More precisely, the partial sums $S_n = \frac{1}{n}\sum_i^n f(x_i) \to \mathbb{E}^\pi[f]$ at a "rate" $\mathcal{O}(\sqrt{N})$.

Definition 1.55 (Metropolis Hastings)**.** *For some function f, not necessarily a probability density, and some "proposal function" g the Metropolis Hastings algorithm on f with proposal g is the following: Initialise some x_0, for each t do the following*

a) Propose some x' according to $g(x'|x_t)$
b) Compute $\alpha = f(x')/f(x_t)$
c) Sample some $u \in [0, 1]$ uniformly
d) If $\alpha > u$ accept x' by setting $x_{t+1} = x'$, otherwise set $x_{t+1} = x_t$

Theorem 1.56. *If the proposal function g is symmetric - that is $g(x|y) = g(y|x)$, and if f is proportional to some probability distribution π, then the Metropolis Hastings algorithm for f with proposal g will generate samples according to π.*

Remark 1.57. If the proposal distribution is not symmetric, a factor known as the Hastings ratio can be introduced to find $\alpha_H = \alpha \cdot \frac{g(x'|x)}{g(x|x')}$ which is used in the acceptance step.

1.3.2.1. *Statistics in MCMC.* While the Metropolis Hastings algorithm (and other MCMC algorithms) converge to π as $n \to \infty$, for finite n the distribution will not necessarily be π, a common example of this is if π is bimodal, where sampling can get "stuck" in one of the modes of the distribution, thereby not fully exploring the space. As such, there are a number of statistics that are used to check if a finite sample from an MCMC algorithm is likely to have converged to the proper distribution π. This is typically done by running multiple independent chains of MCMC.

Definition 1.58 (Variance)**.** *Let there be J chains of L samples, with samples $x_1^j, ..., x_L^j$ for the jth chain. We have the*

a) Between chain variance: $B = \frac{L}{J-1}\sum(\bar{x}_j - \bar{x}_*)^2$
b) Within chain variance: $W = \frac{1}{J}\sum\left(\frac{1}{L-1}\sum(x_i^j - \bar{x}_j)^2\right)$

The between chain variance follows from the central limit theorem that $\mathrm{Var}(\bar{x}_j) \approx \sigma^2 \cdot n$, so $B = \sigma^2 = L \cdot \mathrm{Var}(\bar{x}_j)$, and $\frac{1}{J-1}\sum(\bar{x}_j - \bar{x}_*)^2$ is an unbiased estimate of the variance. The within chain variance is simply the average unbiased estimate of the variance of each chain. From these we can derive

Theorem 1.59 (Gelman-Rubin Variance Estimate)**.** *Let there be J chains of L samples, with samples $x_1^j, ..., x_L^j$ for the jth chain. We have that the total variance is*

$$\hat{V} = \frac{L}{L-1}W + \frac{1}{L}B$$

This result follows directly from the law of total variance.

From this we can construct

Definition 1.60 (Gelman-Rubin Statistic)**.** *Let $\hat{V}$ be the Gelman-Rubin variance estimate, and W be the within chain variance. The Gelman-Rubin statistic is*

$$\hat{R} = \sqrt{\frac{\hat{V}}{W}}$$

When the chains have converged, each chain is a valid sample from π, as such $W = \hat{V}$ and $\hat{R} = 1$. However, if one or more of the chains has not suitably explored the space, $W < \hat{V}$ and $\hat{R} > 1$. The samples will never be a complete representation of the distribution, so typically a threshold of $\hat{R} = 1.01$ is used to indicate convergence.

1.3.3. Simulated Annealing. For a measurable space $(\mathcal{X}, \mathcal{F}, \mu)$ with base measure μ (typically the Lebesgue, or count measure), we may have some measurable function $E : \mathcal{X} \to \mathbb{R}$ known as the energy. In thermal physics, for a thermodynamic system at temperature T, the probability of finding a particle in a given energy E is given by the Gibbs distribution $\pi_T(A) =$

$\frac{1}{Z}\int_A e^{-E(A)/T}d\mu(x)$ where $Z = \int_{\mathcal{X}} e^{-E(A)/T}d\mu(x)$ is a normalising factor. Typically as a system "cools down" it "finds its way" to low energies. Simulated annealing is inspired by this thermodynamic concept, simulating a particle over time as the temperature $T \to 0$ heuristically conjecturing that it will find its way to the lowest energy state. The simulation at a given temperature T is typically done using the Metropolis Hastings algorithm.

While the concept above is inspired as a Heuristic from concepts in thermal physics, it has valid grounding in probability and machine learning. At a high level, this is because for any $x, x' \in \mathcal{X}$, $\pi_T(x')/\pi_T(x) = e^{-(E(x')-E(x))/T}$, which if $E(x') > E(x)$ tends to 0 as $T \to 0$, but if $E(x') < E(x)$ tends to ∞. As such, a sharp delta function is formed around $x_{min} = \arg\min(E)$. While this does mean that at $T = 0$ we have that $\pi(x) = \delta(x - x_{min})$ is the stationary solution to our Metropolis Hastings Markov Transition Kernel, at $T = 0$ the measure of valid starting points $\{x_0\}$ tends to have measure 1 under π but measure 0 under ν. Informally, at low T we almost always accept points where $E(x') < E(x)$ and almost never accept points where $E(x') > E(x)$, so this is just hill climbing and unless E is convex, the sampling will likely get stuck in a local minima.

1.3.4. Transformers.

Definition 1.61 (Row Operator)**.** *For a field F, and a function $s : F^n \to F^n$ define the row operator $\mathcal{R}_s : M_n(F) \to M_n(F)$ as the act of applying s to each row of $M_n(F)$. That is*

$$\mathcal{R}_s\left((r_1, r_2, ..., r_n)^T\right) = (s(r_1), s(r_2), ..., s(r_n))^T$$

Definition 1.62 (Abstract Transformer Head)**.** *For a collection of vectors $(v_1, v_2, ..., v_n)$ where $v_i \in V$ an vector space over a field F. Form $\dot{V}$ by equipping V with a bilinear form denoted $B : V \times V \to F$. Let $G : F^n \otimes V \to M_n(F)$ denote the "Gram operator". Let $L : V \to W$ be an arbitrary linear operator, and $s : F^n \to F^n$ an arbitrary function.*

An abstract transformer head is a function $H : F^n \otimes V \to F^n \otimes W$ defined by

$$H(x) = (\mathcal{R}_f(G(x)) \otimes L)(x) \tag{1.63}$$

Definition 1.64 (Abstract Multi-Head Attention)**.** *For a collection of vectors $(v_1, v_2, ..., v_n)$ where $v_i \in V$ an vector space over a field F. Form spaces $V_1, V_2, ..., V_q$ from V by equipping V with a, possibly unique, inner product. Let $\iota_i : V \to V_i$ be the operation of endowing V with the inner product structure of V_i. Let H_i denote the transformer head from $F^n \otimes V_i \to F^n \otimes W_i$ where W_i, W_j may be distinct. Define $O : W_0 \oplus W_1 \oplus \cdots \oplus W_q \to V$*

The multi-head attention is an operation $A : F^n \otimes V \to F^n \otimes V$ defined as

$$A(x) = (I \otimes O)\left(\bigoplus_i H_i(\iota_i(x))\right) \tag{1.65}$$

Definition 1.66 (Abstract Feedforward Neural Network)**.** *For a vector space U over a field F, a Feedforward is a operator $N : U \to W$ for a pair of linear operators $L_1 : U \to V$ and $L_2 : V \to W$ and a non-linear function $a : V \to V$ is $N(x) = L_2(a(L_1(x)))$*

Definition 1.67 (Abstract Transformer Block)**.** *For a collection of vectors $(v_1, v_2, ..., v_n)$ where $v_i \in V$ an vector space over a field F. A transformer block $T : F^n \otimes V \to F^n \otimes V$ is an operator defined as*

$$T(x) = x + A(x) + (I \otimes N)(x + A(x))$$

While these abstract formulations of the Transformer are theocratically rich, in practice we are working over $F = \mathbb{R}$ and our vector space V is finite dimensional. This allows us to write $V = \mathbb{R}^\kappa$, write all the linear transformations as matrix multiplications, and write the bilinear forms as $B(v_1, v_2) = v_2^T M V_1$ for some matrix M. Also, the non-linear functions used such as f in the abstract transformer head, and a in the feedforward neural network are fixed, or depend on some small set of learnable parameters. This gives a finite set of scaler parameters θ that defines the entire transformer block. Additionally, some further components are introduced into the transformer block to ensure computationally stability and regularisation. This lets us define

Definition 1.68 (Standard Transformer Head)**.** *For a collection of vectors $(v_1, v_2, ..., v_n)$ where $v_i \in \mathbb{R}^d$ we construct $X \in \mathbb{R}^{n \times d}$ by stacking these vectors up as columns. The head is calculated as*

$$h(X) = \text{softmax}\left(\frac{(XW^Q)(XW^k)}{\sqrt{d_k}} + M\right)(ZW^V) \tag{1.69}$$

Where the softmax *is applied along each row. The $W^Q, W^K, W^V \in \mathbb{R}^{n \times d_k}$ are called the "query", "key", and "value" weights. $M \in M_n(\mathbb{R} \cup \{-\infty, \infty\})$ is called the causal mask.*

Definition 1.70 (Standard Multi-Head Attention)**.** *For X defined as above*

$$MHA(X) = \text{Concat}(h_1(X), ..., h_h(X))W^O \tag{1.71}$$

Definition 1.72 (Standard Feedforward Neural Network)**.** *For $x \in \mathbb{R}^\kappa$, $FFN(x) = \sigma(xW_1 + b_1)W_2 + b_2$ where the W_i are called weights and the b_i are called biases. σ is a parameter-less "activation function".*

Definition 1.73 (Layer Norm)**.** *For a vector $x \in \mathbb{R}^d$ define μ and σ as the mean standard deviation of x. The layer norm of x is*

$$LN(x)_i = \gamma_i \cdot \left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\right) + \beta_i \tag{1.74}$$

Where $\gamma, \beta \in \mathbb{R}^n$ are learned parameters, and ϵ is some fixed number for numerical stability.

Remark 1.75. This was not included in the abstract definition of the transformer block as it is primarily used for numerical stability.

Definition 1.76 (Standard Transformer Block)**.** *For X as defined above*

$$Y = X + MHA(LN(X)) + FFN(LN(X + MHA(LN(X)))) \tag{1.77}$$

Where Y is the result from the transformer block.

Because these transformer blocks map from $F^n \otimes V \to F^n \otimes V$, or from $\mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$, they can be chained together.

For many tasks which transformers excel in (language modelling, time series forecasting) the input sequence is sequential, and often not encoded into a vector in a meaningful way. As such we have

Definition 1.78 (Token Embedding)**.** *For an input vectors $v \in \mathbb{R}^V$, the token embedding of v is $h = x^T E$ where $E \in \mathbb{R}^{V\times d}$.*

Remark 1.79. For tasks where the input is categorical, v is usually formed with some one-hot-encoding, that is if there are V categories we define for category c $v_c = (0, ..., 0, 1, 0, ..., 0)$ as a vector in $\mathbb{R}^n$ that has a 1 in the position corresponding to category c (arbitrary), and zeros elsewhere.

Often (e.g. language modelling) there are many categories, so $d < V$ and the embedding becomes a useful technique for encoding "semantic" meaning into a vector.

Once we have embedded our vectors, there is still often a sequential order to them, i.e $(v_0, v_1, v_2, ...)$. The transformer block as described above, is invariant under changing of this order. As such, we use

Definition 1.80 (Positional Encoding)**.** *For a sequence of vectors indexed by time, $(v_0, v_1, ..., v_n)$ each $v_n \in \mathbb{R}^d$. For T_{max} the longest possible sequence, and a positional embedding matrix $P \in \mathbb{R}^{T_{max}\times d}$, the positional embedding of $(v_0, v_1, ..., v_n)$ is defined with $v_i \mapsto v_i + P_i$ where P_i is the ith row.*

A prototypical example of applying these steps is in GPT-2, described as follows. For a sequence of words $(w_1, w_2, ..., w_n)$, each word in a vocabulary of size V, encode each word via one hot encoding. Perform token embedding into $\mathbb{R}^d$, and perform a positional encoding. Apply a series of transformer blocks sequentially, apply a single layer norm, apply a linear projection back into $\mathbb{R}^V$ for each "word" (via some matrix multiplication), and finally apply softmax to each output "word". The final vector is trained to represent the probability of the next word in the sequence of words. So for position 1 the prediction is for w_2, for position n the prediction is for word w_{n+1} not in the data. To train GPT-2 we use an objective function that calculates the cross entropy between the prediction at each step and the actual next word, this is done for each "word" in the output layer.

1.3.5. Gradient Descent. If we have a neural network $N(x \mid \theta)$ that takes in some input $x \in V$ and some parameter set $\theta \in \mathbb{R}^n$ and produces some output $y \in W$ where V, W are vector spaces over a field F, typically $\mathbb{R}$, and typically finite dimensional. We typically have some "training

data", which is a set of $\{(x_i, y_i) \mid x_i \in V, y_i \in W\}$ where each y_i is the output that we "want" from $N(x_i, \theta)$, what we mean by "want" is that some function $\mathbb{E}_{x_i}[\mathcal{L}(N(x_i, \hat{\theta}), y_i)]$ is minimised by our parameter choice $\hat{\theta}$. To find this target $\hat{\theta}$. Because N is typically complicated, and has a complex dependence on θ, we cannot directly minimise this objective function to find $\hat{\theta}$, as such we use gradient decent. The concept behind gradient decent is to take some θ_i, find a local, linear approximation of $\mathbb{E}_{x_i}[\mathcal{L}(N(x_i, \hat{\theta}_i), y_i)]$, and find some "penalty" for "trusting" the approximation too much. Precisely, we compute

$$\theta_{i+1} = \arg\min_{\theta} \left(\langle g^T, (\theta - \theta_i) \rangle + \frac{\lambda}{2} \|\theta - \theta_i\|^2 \right) \tag{1.81}$$

This is done recursively, typically for some number of steps. Here g is the gradient, often we would consider using $\nabla_\theta \mathbb{E}_{x_i}[\mathcal{L}(N(x_i, \hat{\theta}_i), y_i)]$, but computing this over the entire "training" data set is slow, so we typically a small "batch" of data is taken from the training data and the gradient is taken over this batch. Because this batch is "random" a small subset of the true data, our estimate of the gradient is going to be somewhat stochastic, as such many gradient decent algorithms use some sort of exponential smoothing step for the gradient - this is often called the momentum. Different choices of exponential smoothing, and different choices of the norm, and different choices of the gradient lead to different types of optimisers. A few common ones are *SGD* which uses the frobenius norm, the standard gradient, and no smoothing; *adam*, which uses the $\ell_1 \to \ell_\infty$ norm, the standard gradient, and exponential smoothing; *shampoo*, which uses the spectral norm, the standard gradient, and exponential smoothing; and *NGF* which uses the frobenius norm, the natural gradient, and no exponential smoothing.

1.3.6. Reinforcement Learning. For our specific usecase, reinforcement learning is a technique used when we have some "reward function" $R(y) : Y \to \mathbb{R}$ that tells us how good a specific sample is, and we want to update the parameters of some probability distribution / sampling function $Y \sim \pi_\theta$ to maximise the expected reward, that is

$$J(\theta) = \mathbb{E}_{y \sim \pi_\theta}[R(y)] \tag{1.82}$$

While reinforcement learning is typically more general than this, and applies to policy learning, this will suffice for our purpose.

The most direct way to update π_θ is to simply take a number of samples from π_θ and calculate

$$g = \nabla_\theta J(\theta) = \mathbb{E}_{y \sim \pi_\theta}[\nabla_\theta \ln(\pi_\theta) R(y)] \tag{1.83}$$

and then use this g in gradient decent as described above.

While the above techniques makes logical sense, the use of $J(\theta)$ being the expected reward can become problematic, for example if our reward function is binary, returning a score of 1 if

the output meets some criteria, and returning a score of -1 if it does not, once the parameter set has settled into a stable configuration where each output is valid and produces a score of 1, $J(\theta)$ continues to update θ which can lead to oscillations. As such we often use an "advantage" function A which is an estimate of how the score $R(y_i)$ for a specific sample compares to the expected score. That is $A(y_i) = R(y_i) - \mathbb{E}_{y\sim\pi_\theta}[R(y)]$. We can estimate the expected score by simply sampling from π_θ a few times, and taking the average, and for a simple problem this is sufficient.

Remark 1.84. This average often has a high variance, so a "critic" model is often trained instead to try and learn the baseline. We won't explore this technique here.

We can use the advantage to get a refined objective function $J(\theta) = \mathbb{E}_{y\sim\pi_\theta}[A(y)]$. We can calculate the gradient once again as above, and apply gradient accent. The primary disadvantage here is that each time we take a gradient step, we have to resample from π_θ to get a new batch to calculate the new gradient. The new distribution with parameter set θ will only be slightly different to the original parameter set before the gradient update θ_{old}, and as such we can keep the old samples $y_i \sim \pi_{\theta_{old}}$ and then use a technique called importance weighting to find that

$$\mathbb{E}_{y\sim\pi_\theta}[A(y)] = \mathbb{E}_{y\sim\pi_{\theta_{old}}}[r(\theta)A(y)] \tag{1.85}$$

Where $r(\theta) = \frac{\pi_\theta(y)}{\pi_{\theta_{old}}(y)}$ is called the importance ratio. This allows us to take batch of samples from $\pi_{\theta_{old}}$, perform a few steps of gradient accent using this batch, importance reweighing at each step. While this works in expectations, typically after a few gradient steps our samples are no longer representative, and our reweighing erodes. Therefore, it is typical to use a "trust region" wherein we can trust an update. One of the most common ways to do this is to "cap" positive updates by restricting $r(\theta) \leq 1 + \epsilon$ if $A(y) > 0$, that is if if we generated good samples from θ_{old} we want to increase the likelihood of generating them, but not so much so that we can no longer trust the importance sampling as the ratio $r(\theta)$ has become too big, but on the other hand, if the advantage $A < 0$, we want to continue to have a learning signal. This yields the proximal policy optimisation (PPO) objective

$$L^{CLIP}(\theta) = \mathbb{E}_{y\sim\pi_{\theta_{old}}}[\min\left(r(\theta)A, \text{clip}(r, 1-\epsilon, 1+\epsilon)A\right)] \tag{1.86}$$

CHAPTER 2

Problem Formulation

We begin by simply stating the problem that we are studying. The central problem is the optimisation of an objective function over the space of manifold triangulations. Formally given the space of all triangulations of some manifold M, denoted $\mathcal{T}(M)$, and some *objective function* $\mathcal{O} : \mathcal{T}(M) \to \mathbb{R}$ (that is some function we are interested in maximising) we want to find some $T^* \in \mathcal{T}(M)$ subject to some specific size constraint that maximises this function:

$$T^* = \arg \max_{T \in \mathcal{T}(M), \text{size}(T) \leq S_{\max}} \mathcal{O}(T) \tag{2.1}$$

Because the space $\mathcal{T}(M)$ is intractably large and poorly understood, finding the global maxima is likely impossible, as are any analytic techniques. Hence, we develop and compare a series of techniques that can be used to find a specific triangulation $T \in \mathcal{T}(M)$ with $\mathcal{O}(T)$ significantly higher than a typical triangulation from $\mathcal{T}(M)$.

To achieve this we address two key sub-problems

a) **Establishing a Baseline:** We need to explore techniques of quantifying for a specific triangulation $T \in \mathcal{T}(M)$ what it means to be significantly better than a typical triangulation in $\mathcal{T}(M)$.

b) **Generative Modelling:** We lay the groundwork for a range of new optimisation techniques from the field of reinforcement learning. This involves first establishing a technique that can be used to generate triangulations from $\mathcal{T}(M)$ using deep learning techniques.

The motivation for this work stems from a range of fields where optimising some objective functions over $\mathcal{T}(M)$ is of particular interest. While the techniques we develop are specifically designed to be agnostic to the objective function, we focus primarily on one specific class of objective functions that relate to the knots present in triangulations as a proof of concept.

CHAPTER 3

Search and Sampling Algorithms

3.1. Classical Techniques

3.1.1. Direct Accent. For an objective function $O : \mathcal{T}(\mathcal{M}) \to \mathbb{R}$, for small triangulations (number of tetrahedra around 6 or lower), we can typically calculate O exhaustively from a census. In doing this, we may find that O tends to increase with more tetrahedra, this motivates the notion of direct accent. The concept is to start with some some small triangulation, enumerate all of its neighbours that have more tetrahedra, and then sample one of these randomly, biased towards choosing one with a higher objective. This process can then be repeated to generate a chain of triangulations of increasing size, and the whole process repeated to generate multiple chains.

In order for this search technique to work, we need to be able to do two things.

a) enumerate all the neighbours of a specific triangulation T' that are larger, call this set $N_+(T')$
b) convert from $\{\mathcal{O}(T) \ : \ T \in N_+(T')\}$ to some selection probability

To address this first point, we consider $(2-3)$ moves. In order for a $(2-3)$ move to happen we need to have a face that is shared by two distinct tetrahedra, this face will be converted to an edge that is "perpendicular" in some sense. Hence, we simply iterate through all $2T$ faces and check if a $(2-3)$ move is possible, performing it if it is. The details are explained in more detail in Altmann and Spreer [**?**].

For the second point, we need to convert from a list of scores $\{O_1, ..., O_n\}$ to a list of probabilities $\{p_1, ..., p_n\}$ ensuring that the probability is positive and normalised to sum to one. This is a standard problem in machine learning and is addressed by the softmax function

$$p_i = \frac{e^{\beta O_i}}{\sum_j e^{\beta O_j}} \tag{3.1}$$

Where $\beta > 0$ is some fixed parameter, often related to the inverse of the "temperature" $\beta = 1/T$, lending from an analogy to thermal physics.

With these defined, we can now detail direct ascent in algorithm 1. The algorithm details a single run of direct ascent, this can be done multiple times to get a range of different ascent paths. So long as $\beta < \infty$ there is a possibility for unique paths. When $\beta = 0$ it is essentially a random search, choosing any neighbour independent of the objective functions value. When $\beta = \infty$ or is particularly large, the neighbour chosen is the one with the greatest objective of all neighbours.

Algorithm 1 Direct Accent

Let $T_0 \in \mathcal{T}(M)$ be the initial triangulation.
Let $\beta > 0$ be a fixed parameter.
let $N \in \mathbb{N}$ be a fixed chain size.
Let $\mathcal{O} : \mathcal{T}(M) \to \mathbb{R}$ be some objective function.
for $n = 1$ to N **do**
  Generate all neighbours T_{ni} of T_{n-1} by applying $(2-3)$ moves.
  for each neighbours T_{ni} in the set **do**
    Calculate the selection probability $p_i = \frac{e^{\beta O_i}}{\sum_j e^{\beta O_j}}$.
  end for
  Sample T_n from the set of neighbours $\{T_{ni}\}$ with probability p_i.
end for
Return T_N

3.1.2. Markov Chain Monte Carlo. For an objective function $\mathcal{O} : \mathcal{T}(M) \to \mathbb{R}$, we will be interested in knowing what a "typical" objective function value looks like. One way to do this is to adapt Markov chain Monte Carlo (MCMC) to the Pachner graph, and generate a uniform sampling of the space, calculating $\mathcal{O}$ on each member of the sample and analysing its statistical properties. The main issue with this is that the Pachner graph is infinite, so its impossible to define a uniform distribution on the space. Furthermore, the number of triangulations for a given number of tetrahedra increases incredibly quickly, and at an unknown rate. Hence, a uniform sampling across all levels would be currently impossible to prove (as the number of triangulations is unknown) and would spend "more time" at large triangulations. Hence, we modify the problem to the goal of returning a sample that is uniform for a given level, but is otherwise different for different levels, and in particular the probability of sampling a triangulation of size t decreases as t gets larger *faster* than the number of triangulations grows, this ensures that the probability of sampling large triangulations tends to zero, moderating the size of the triangulations sampled. The specific algorithm developed to do this was developed by Altmann and Spreeer. The algorithm is outlined in 2. We note that $n(T_s)$ is the number of tetrahedra of T_s and an $i-$neighbour is a triangulation resulting from an $i-$Pachner move for $i = 2, 3$ is a $(3-2)$ and $(2-3)$ move respectively. This algorithm represents a single chain of MCMC, typically multiple are run to ensure convergence as in accordance with definition 1.60

3.1.3. Simulated Annealing. Because the above Markov Chain Monte Carlo algorithm leads to a uniform distribution, we can use it as a proposal distribution for simulated annealing. This allows us to directly modify the MCMC algorithm to become a simulated annealing algorithm. The concept is to propose a triangulation, compare its objective function to the current objective function, if it is better move to this new triangulation, if it is worse move to the new triangulation with some probability, staying at the same triangulation otherwise. Similar to direct ascent, we need to define some way of calculating the probability of moving in the scenario that it is worse,

Algorithm 2 MCMC for Triangulations

Let $T_1 \in \mathcal{T}(M)$ be the initial triangulation.
Let $\gamma = 1/k, k \in \mathbb{N}$
Let N be the number of iterations
Let $\mathcal{O} : \mathcal{T}(M) \to \mathbb{R}$ be some objective function.
for $s = 1$ to S **do**
    Sample $u \in U([0, 1])$ uniform
    if $u < e^{-\gamma n(T_s)}$ **then**
        Set $i := 1$, $m := 2n$
    else
        Set $i := 2$, $m := 2n - 2$
    end if
    Enumerate $i-$neighbours $T'_1, ..., T'_\ell$ of T_s
    Sample $v \in U([0, 1])$ uniform
    if $v < \ell/m$ **then**
        Set T_{s+1} to a random choice of $T'_1, ..., T'_\ell$
    else
        Set T_{s+1} to T_s
    end if
end for

and similar to direct ascent we chose a function

$$p(o_p, o_c) = e^{-\beta(o_c - o_p)} \tag{3.2}$$

Where once again $\beta = 1/T$ controls how much randomness is involved. Unlike direct ascent, the value of β is usually chosen to vary from iteration to itinerate. This is called a temperature schedule. So we have β_n for each step n.

The implemented standard simulated aneling algorithm is described in algorithm 3.

The key to this algorithm performing successfully is that we have a good choice for β_n. If it's too high we will get stuck in local maxima, and if its too low we essentially have a random walk which won't achieve much more than MCMC. As such, we pick a "target acceptance rate". The "acceptance rate" is the percentage of the time that we accept a proposal as opposed to staying still. Higher acceptance rates correspond to random walks, and lower ones correspond to the algorithm sitting in the one place. The "target acceptance rate" is what we want the acceptance rate to be. During sampling, we track a exponential moving average estimate for what the current acceptance rate is according to

$$r_n = (1 - \alpha)r_{n-1} + \alpha a_n \tag{3.3}$$

Where $a_n = 0$ if the move was not accepted and $a_n = 1$ if the move was accepted. We then compare the moving average acceptance rate r_n to the target acceptance rate r and compute the

difference $r_n - r$ and update β according to

$$\beta_n = \beta_{n-1} e^{\lambda(r_n - r)} \tag{3.4}$$

Here λ represents how aggressively to track the target, and α represents how quickly to forget past acceptance rates. This gives us an adaptive simulated annealing.

Algorithm 3 Simulated Annealing

Let T_0 be some starting triangulation.
Let N be the chain length.
Let M be some hash table memory.
Let $\mathcal{O} : \mathcal{T}(M) \to \mathbb{R}$ be some objective function.
Let β_n be some temperature schedule.
for $n = 1$ to N **do**
    Propose T'_n from T_{n-1} by using $1-$step of the described MCMC algorithm.
    Retrieve $o_{n-1} = \mathcal{O}(T_{n-1})$ from M
    Check if $o_n = \mathcal{O}(T_n)$ is in M, if it is - retrieve it, if not - compute it and store it in M.
    Compute $\alpha = e^{-\beta_n(o_{n-1} - o_n)}$
    Generate some $p \sim U([0, 1])$
    if $p \leq \alpha$ **then**
        Accept $T_n = T'_n$
    else
        Let $T_n = T_{n-1}$
    end if
end for

3.2. Transformers

3.2.1. Base Transformer. We use a standard GPT-2 style transformer with dropout and pre-layer normalisation. The final transformer model takes the sequence, performs token and positional embedding, dropout, and then n_{layers} of the standard transformer block as described in definition 1.3.4. Between each transformer block there is a dropout of 0.1 applied. This architecture is the standard GPT-2 style architecture. We use the same dimension for the embedding dimension and the feed forward neural networks. The vocabulary size includes all the letters of the training data, plus three special tokens `[BOS]`, `[EOS]`, and `[PAD]` for the beginning of sentence, end of sentence, and padding respectively. For training we use AdamW, with a cross entropy objective function.

CHAPTER 4

Numerical Experiments

4.1. Objective Functions

To test the effectiveness of our optimisation strategies on triangulations of manifolds, we consider 5 specific objective functions. These objective functions are all considered on single vertex triangulations of S^3, call the set of all of these $\mathcal{T}_1(S^3)$.

4.1.1. Alexander Polynomial. For any $T \in \mathcal{T}_1(S^3)$, we can consider an edge e_α, because the triangulation has only one vertex, e_α forms a loop. This loop can be knotted in S^3. Call the associated knot for e_α K_α. The Alexander polynomial $\Delta_{K_\alpha}(t)$ has coefficient vector $\vec{a} = [a_0, a_1, ..., a_n]$. It is known that "more complex" Alexander polynomials lead to "more complex" knots. This notion suggests 3 objective functions of interest.

a) $\mathcal{O}_{deg} : T \mapsto \sum_{e_\alpha} \deg(\Delta_{K_\alpha}(t))$ where because the Alexander polynomial is invariant under multiplication by $\pm t$ we use the difference between the highest and lowest power of t.
b) $\mathcal{O}_{det} : T \mapsto \sum_{e_\alpha} |\Delta_{K_\alpha}(-1)|$.
c) $\mathcal{O}_{norm} : T \mapsto \sum_{e_\alpha} \|\vec{a}\|^2$ where $\vec{a}$ is the coefficient vector of the Alexander polynomial associated with the knot formed by the edge e_α in S^3

The first of these objective functions, $\mathcal{O}_{deg}$ is motivated by the fact that $2g(K) \geq \deg(\Delta_K(t))$. The second objective function, $\mathcal{O}_{det}$ is motivated by the connection between the determinant $|\Delta_K(-1)|$ and the knot colouring. The third objective function, $\mathcal{O}_{norm}$ is a standard objective function on vectors.

4.1.2. Fundamental Group of $T\backslash\nu(e_\alpha)$. As discussed, for $T \in \mathcal{T}_1(S^3)$, any edge e_α can form a knot. Typically in knot theory we are interested in the study of the knot complement $M_\alpha = S^3\backslash\nu(K_\alpha)$. This inspires our fourth objective function

d) $\mathcal{O}_{gen} : T \mapsto \sum_{e_\alpha} \#_{gen}(\pi_1(M_\alpha))$ where $\#_{gen}$ is the number of generators of the presentation of the fundamental group.

While we would ideally like to use the reduced presentation of the fundamental group, as discussed in **??** there is no algorithm to do this always in finite time, so we use the standard presentation returned by regina [**?**], as a heuristic, and then a reduction process can be attempted afterwards to check if the fundamental group can be reduced, and if so what it can be reduced to.

4.1.3. Edge Degree Variance. The prior four objective functions are all inspired by the knot structure of our triangulations. While this is of particular interest because it is a topological property

of the triangulation, we would also like to consider the complexity of the triangulation itself. This inspires us to consider the degree of the edges in our manifold, where the degree of an edge is the number of tetrahedra that share this edge.

e) $\mathcal{O}_{var} : T \mapsto \mathrm{Var}_{e_\alpha}(\deg(e_\alpha))$ where we have taken the variance.

The choice of variance here follows from the simple argument that $V = 1$ because we have defined our space of triangulations to be single vertex triangulations. We know that $\chi = 0$ for S^3, so $1 - E + F - T = 0$. Also we know that each of the tetrahedra have 4 faces, but because S^3 is closed they have to be glued together in pairs, so $F = 2T$, this means $E = 1 + T$. Also, each tetrahedra has to contain 6 edges, so the total edge degree must be $6T$, and the average edge degree $\frac{6T}{T+1}$, hence this is constant for a fixed number of tetrahedra, regardless of the triangulation. Hence the variance is chosen because it captures how "uniform" or conversely "irregular" the triangulation is.

4.2. Markov Chain Monte Carlo

To establish a baseline for our objective functions, generic Markov Chain Monte Carlo was run to determine the distribution of our objective function across the Pachner Graph. The MCMC algorithm described by Altmann and Spreer [**?**] samples uniformly around triangulations of a particular number of tetrahedra. We, choose to examine the Pachner graph around triangulations of 30-tetrahedra, which corresponds approximately to $\gamma = 1/10$. This choice is informed by other results in computational topology that have found interesting triangulations of size less than 30 tetrahedra (for example Burton [**?**]), and because 30 tetrahedra is intractable to enumerate exhaustively. Samples with 7 chains of $10,000$ iterations with a step size of 100 where performed, for each sample all objective functions were calculated (as opposed to running distinct sampling processes for each objective function). This has the benefit of not only saving computational time, but giving us a uniform sample of the space where we can asses the correlation of each objective function. We confirmed convergence with the Gelman-Rubin statistic with a threshold of 1.01, this threshold is in accordance with modern recommendations [**?**]. The convergence statistics are in table 1. For interest, figure 1 shows the correlation between each objective functions across the entire set of MCMC samples. It is important to note that this correlation tells us the correlation for "an average" sample, and the correlation between the functions could dramatically change in very specific parts of the distribution - in particular near the respective maxima of each variable.

We extracted the triangulations with 30 tetrahedra, there were $8,250$ such triangulations so an efficiency of $\approx 12\%$. The following analyses are performed on this subset of triangulations.

Metric	$\hat{R}$
$\mathcal{O}_{norm}$	1.003
$\mathcal{O}_{deg}$	1.004
$\mathcal{O}_{det}$	1.005
$\mathcal{O}_{var}$	1.006
$\mathcal{O}_{gen}$	1.009

TABLE 1. Convergence statistics of MCMC Runs

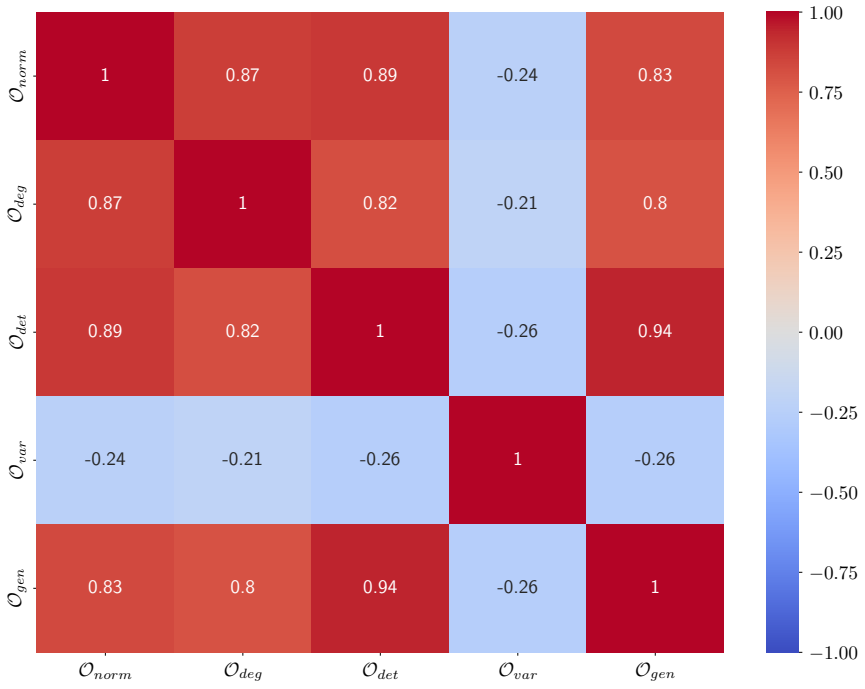
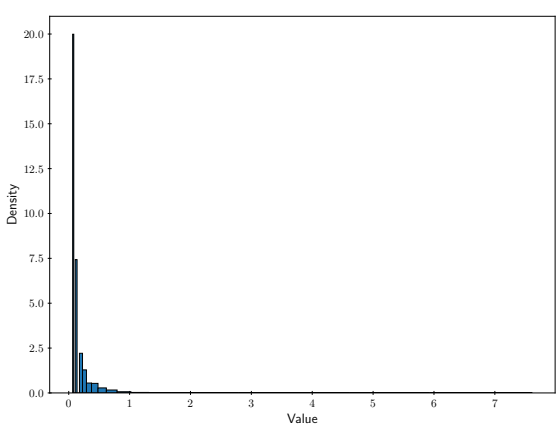


FIGURE 1. Correlation plot for each objective function.

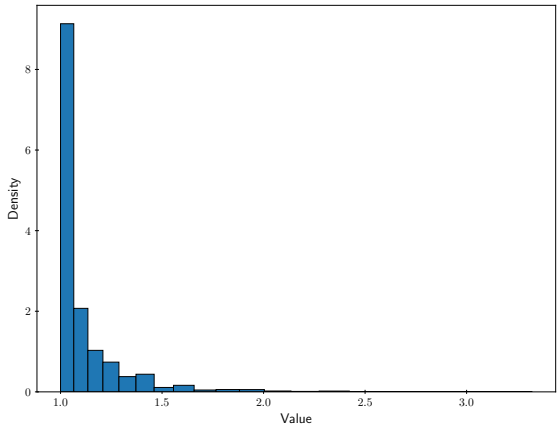
The distribution for the score for each objective function outlined in section 4.1 is displayed in figure 2.

All but $\mathcal{O}_{var}$ present power law distributions or power law distributed tails. To confirm this we fit both a power law distribution across the entire data, and also compute $x_{\min}$ according to the procedure of Clauset, Shalizi, and Newman, and fit a power law distribution to the tails. Both these fits are done with MLE. A plot on a log-log histogram of the data, and the fitted distributions is seen in figure 3.

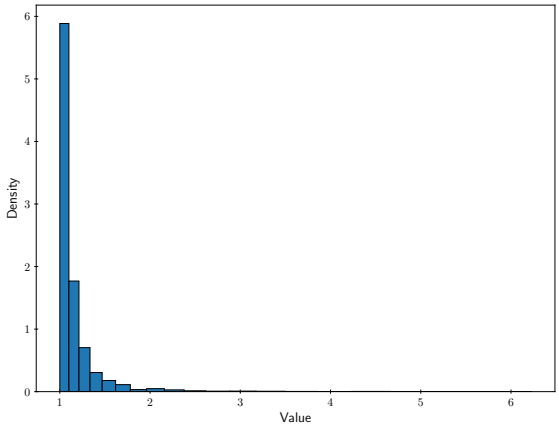
The $\mathcal{O}_{var}$ appears to have some positively skewed distribution, considering that it is the distribution of the variance, a chi squared distribution would be expected. However, fitting one to the data lead to poor results and a log normal distribution was used instead. This is weakly justified, asside from a strong in sample fit as seen in figure 4 that is plotted in log-log space to confirm tail behaviour.



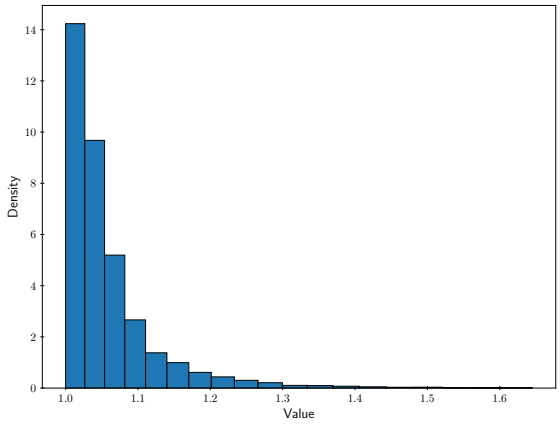
(A) $\mathcal{O}_{deg}$



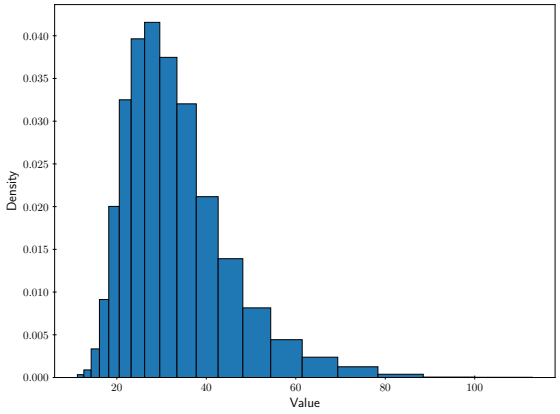
(B) $\mathcal{O}_{det}$



(C) $\mathcal{O}_{norm}$

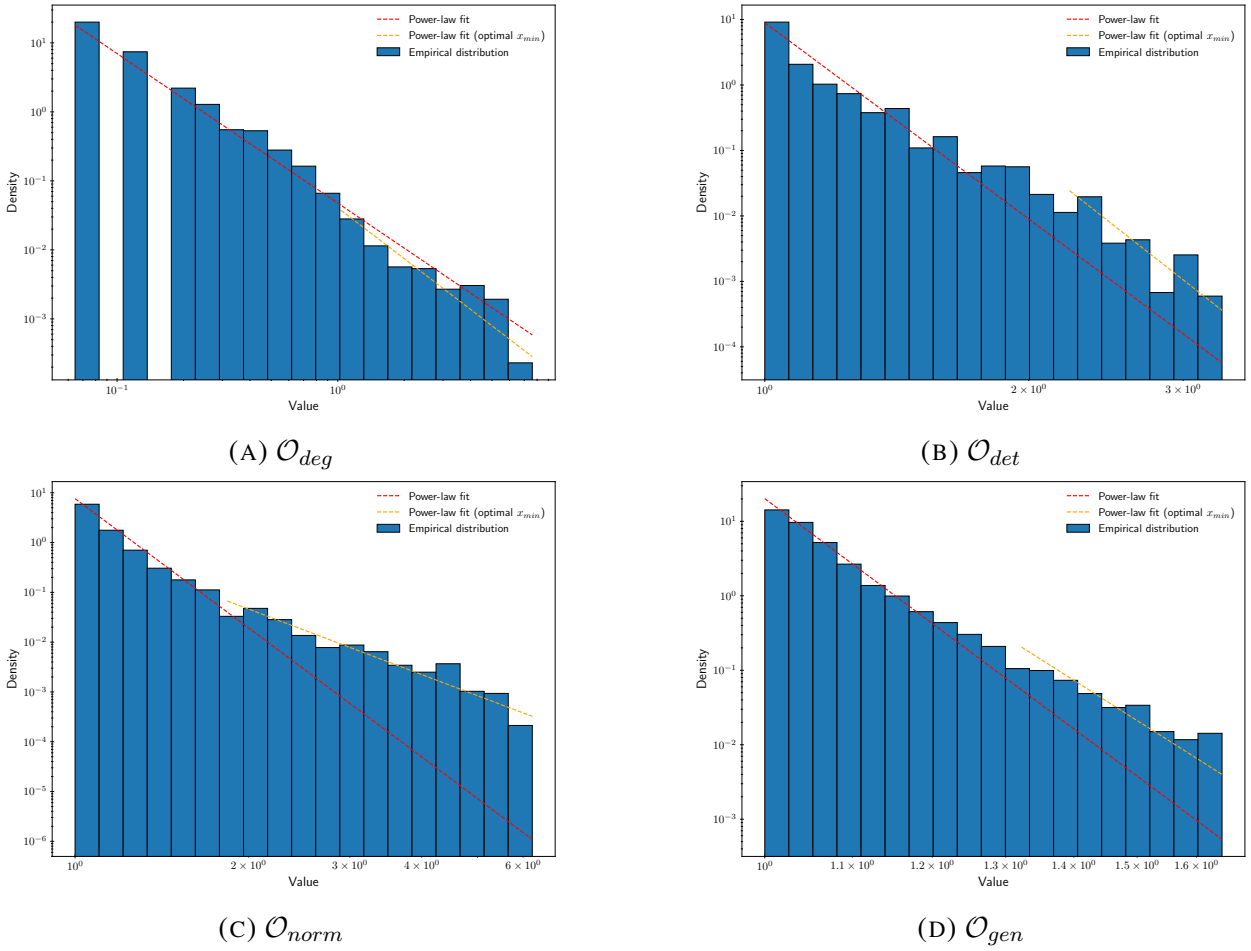


(D) $\mathcal{O}_{gen}$



(E) $\mathcal{O}_{var}$

FIGURE 2. Distribution of the score for each objective function under MCMC samples at $\gamma = 1/10$



(A) $\mathcal{O}_{deg}$ (B) $\mathcal{O}_{det}$

(C) $\mathcal{O}_{norm}$ (D) $\mathcal{O}_{gen}$

FIGURE 3. Log-log histogram of the score for each objective function under MCMC samples at $\gamma = 1/10$. The red line indicates the fitted power law distribution over the entire dataset, the orange line the fitted power law distribution for $x > x_{\min}$

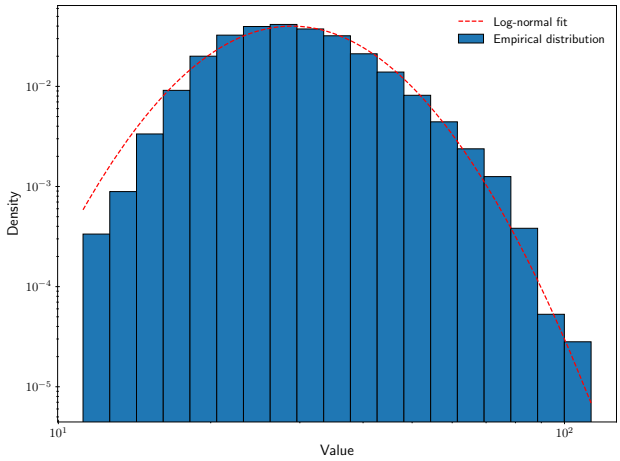


FIGURE 4. Log-log histogram of $\mathcal{O}_{var}$ with fitted log normal distribution.

4.3. Classical Optimisation

4.3.1. Direct Ascent. We perform direct accent as described in section 3.1.1 starting from the seed triangulation of `cMcabbgqs` which has one vertex and two tetrahedra. We performed this direct accent 20 times at different temperatures ($\beta = 1/T = 0.1, \sqrt{0.1}, 1, \sqrt{10}, 10$) and compared the maximum achieved score at each level, an example of this for $\mathcal{O}_{deg}$ in figure 5. In all cases it was found that a lower temperature lead to better results, this suggests that the graph is connected enough that greedy ascent is sufficient (that is, the neighbour with the greatest score is accepted at each step).

We perform greedy ascent for each objective function for 28 steps to a triangulation of 30 tetrahedra, the chains are presented in figure 6.

We see that for each objective function the ascent profile is almost fully deterministic. All objective functions demonstrate monotonic growth. $\mathcal{O}_{deg}$ demonstrates accelerated growth, and $\mathcal{O}_{det}$, $\mathcal{O}_{var}$, and $\mathcal{O}_{norm}$ demonstrate linear growth; in either case this suggests that an arbitrarily high score can be achieved by simply extending this procedure to larger triangulations. On the other hand, $\mathcal{O}_{gen}$ appears to be plateauing suggesting that the exploration technique may not yield significant improvement as we extend to large triangulations. Interestingly, $\mathcal{O}_{var}$ presenting slightly better than expected values for even triangulations than odd triangulations (if we assume linearity), this leads to a slight oscillating behaviour.

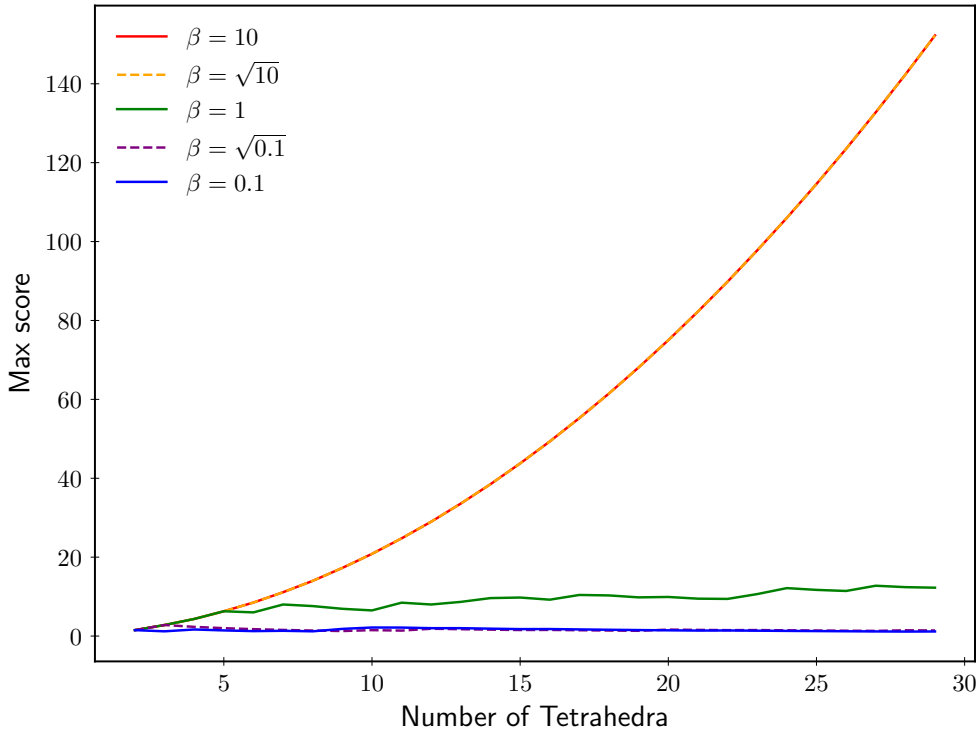
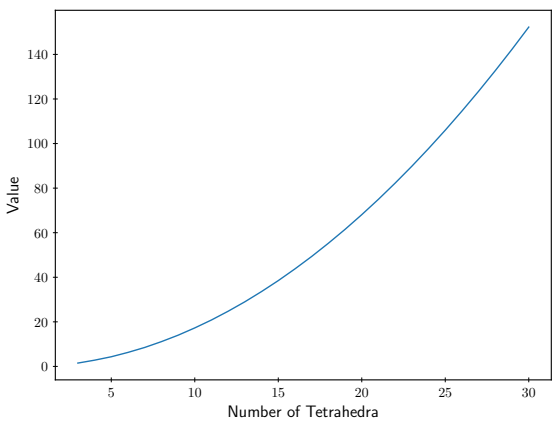
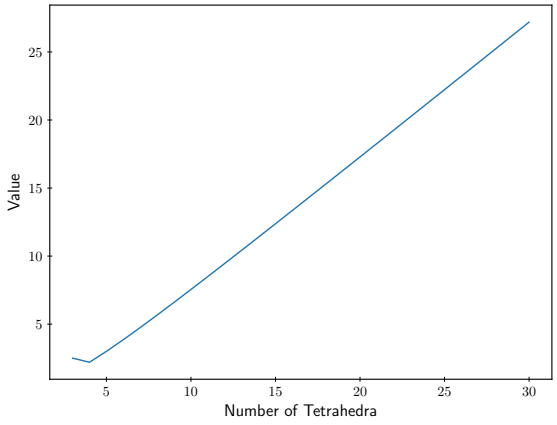


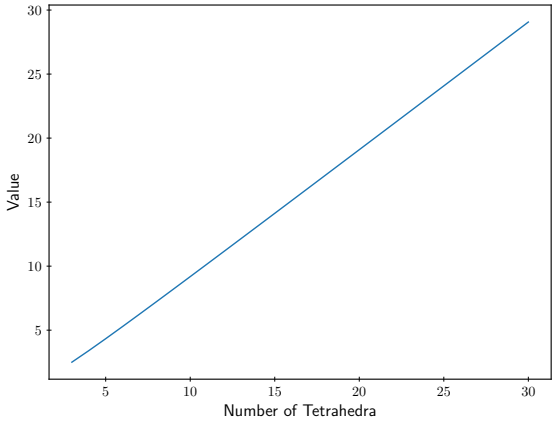
FIGURE 5. Direct Ascent on $\mathcal{O}_{deg}$ at different temperatures ($\beta = 1/T$)



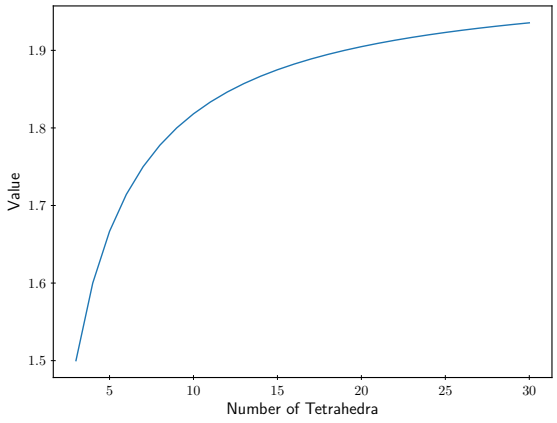
(A) $\mathcal{O}_{deg}$



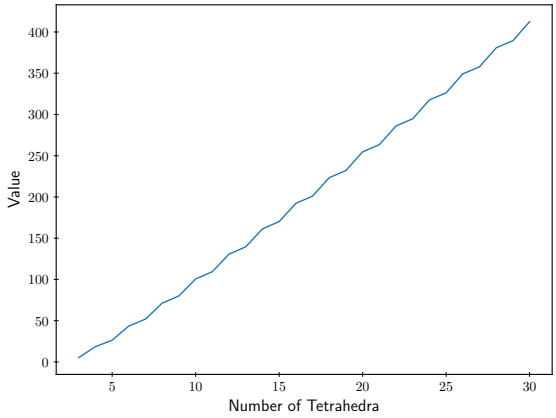
(B) $\mathcal{O}_{det}$



(C) $\mathcal{O}_{norm}$



(D) $\mathcal{O}_{gen}$



(E) $\mathcal{O}_{var}$

FIGURE 6. Direct / greedy ascent for each objective function.

While the direct ascent approach does appear to perform particularly well and predictably, and is an incredibly simple algorithm to implement. It only generates a single triangulation, and requires calculating the objective function on a large number of discarded triangulations - in total to generate the ascent profile of 28 triangulations required exploring on the order of $1,000$ triangulations. As

such, in situations where a single high value triangulation is desired, this technique demonstrates value. However, if the goal is to explore a variety of different high valued triangulations and have a high exploration efficiency - this technique is not suitable.

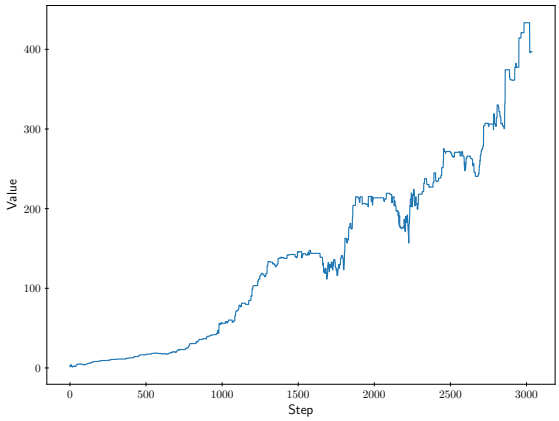
4.3.2. Simulated Annealing. We perform simulated annealing as described in section 3.1.3. This is done for a target of $10,000$ iterations with a step size of 10 and target acceptance rate of 20%, $\alpha = 0.01$ and $\lambda = 0.1$, this acceptance rate was chosen heuristically based on a number of different small experiments. To ensure a fair comparison with direct ascent, the potential was set to $-\infty$ for triangulations with more than 30 tetrahedra. For $\mathcal{O}_{deg}$ and $\mathcal{O}_{norm}$, the chain was stopped early at $3,307$ and $2,940$ samples because the computational time required to score a single triangulation had become unfeasibly slow (> 10second$/$sample). The actual acceptance rate was calculated in each case. For $\mathcal{O}_{det}, \mathcal{O}_{gen}$ and $\mathcal{O}_{var}$ the achieved acceptance rate was within 1% of the target. For $\mathcal{O}_{deg}$ and $\mathcal{O}_{norm}$ the acceptance rate is lower, though this is likely due to the reduced number of samples achieved. The chains for each objective function and the achieved acceptance rates are included in figure 7.

For $\mathcal{O}_{deg}$ and $\mathcal{O}_{norm}$ which were both stopped early, the objective function appears to still be increasing, suggesting that a higher objective function could have been achieved if let run for the full 10,000 iterations. However, it is important to realise that the more complex a knot is, the longer it takes to calculate the Alexander polynomial, so if the algorithm was to continue to run and improve the objective function, it is likely that it would continue to slow down. The computation for the Alexander polynomial is slow because the symbolic determinant of a large collection of matrices is required to be calculated. This is a poorly optimised problem. However, the full symbolic result is possibly not required, and specific algorithms to calculate the degree and norm could potentially be developed to significantly improve the compute time, and leverage the parallelism of GPUs. $\mathcal{O}_{gen}$, and $\mathcal{O}_{var}$ both appeared to plateau relatively early and did not show significant improvement after around $2,000$ epochs. Similar could be suggested for $\mathcal{O}_{det}$, though the results are less conclusive.

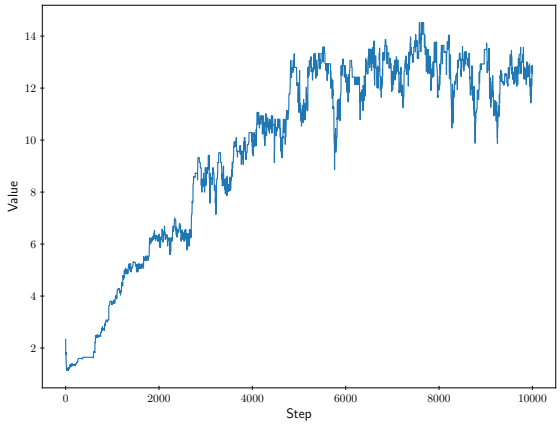
4.4. Comparison of Classical Optimisation

The best result from each optimisation technique are compiled in table 2. This table includes the achieved percentile of the best sample, computed by the fitted power law distribution for both the entire distribution (overall) and the tail distribution (tail) from section 4.2.

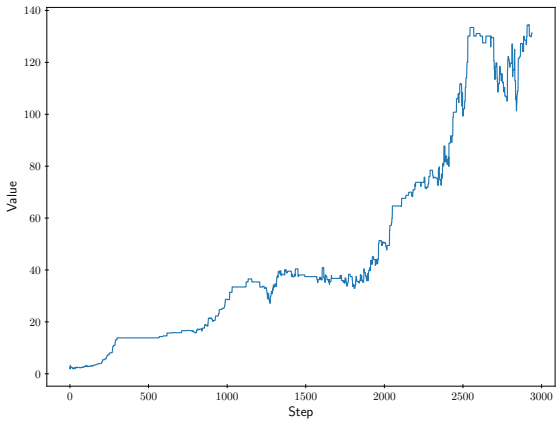
In all cases the search techniques uncovered samples that where at least in the 10^{-5}th percentile. Meaning that $100,000$ samples of 30 tetrahedra would be required. Given the 30 tetrahedra sampling rate was on the order of 10% at least $1,000,000$ samples from MCMC would be required to find a solution as good, simulated annealing only used $10,000$, and direct ascent only around $1,000$ making the classical search techniques at least $100\times$ more efficient. In the best case scenario, the



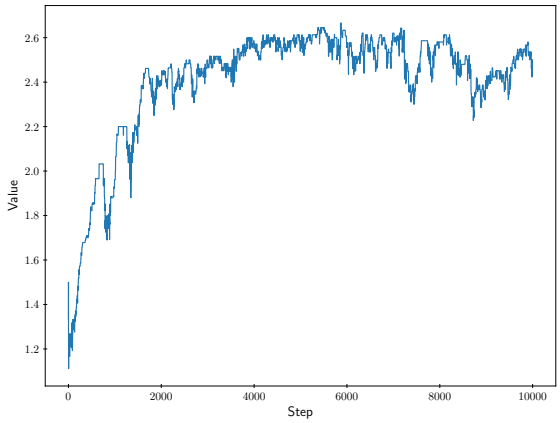
(A) $\mathcal{O}_{deg}$ acceptance rate: 17.78%



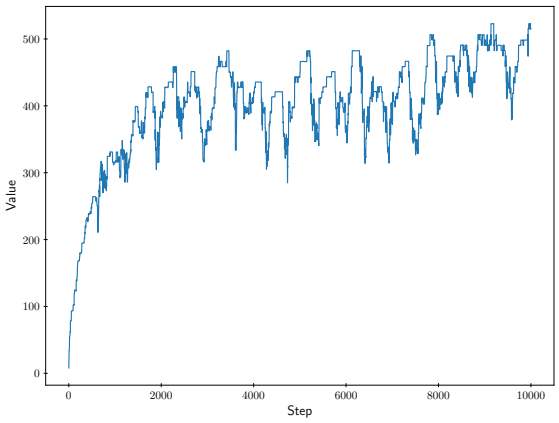
(B) $\mathcal{O}_{det}$ acceptance rate: 19.96%



(C) $\mathcal{O}_{norm}$ acceptance rate: 18.44%



(D) $\mathcal{O}_{gen}$ acceptance rate: 20.11%



(E) $\mathcal{O}_{var}$ acceptance rate: 19.44%

FIGURE 7. Trace plots for simulated annealing on each objective function for 10,000 iterations and a 20% acceptance rate.

score was in the 10^{-13}th percentile (conservatively choosing the largest of total and tail) making the search at least $10^{10}\times$ more efficient.

With the exception of $\mathcal{O}_{det}$, the simulated annealing performed better than the greedy ascent. That being said, simulated annealing also had to perform $2\times$ to $5\times$ more calculations than the direct ascent (of the $10,000$ steps, the model would sometimes propose an already visited node which didn't need to be recomputed). The improvement was 1-2 orders of magnitude in percentile (conservatively) suggesting that simulated annealing is the better choice when maximising the objective function as much as possible is desired. Though the increased complexity in the algorithm, and the slower $2\times$ to $5\times$ slower runtime are drawbacks if absolute maximums are not required. Interestingly, $\mathcal{O}_{det}$ performed significantly better under direct ascent. Achieving a particularly high percentile on the order 10^{-13}. It was not entirely clear that simulated annealing for $\mathcal{O}_{det}$ had settled down, and a longer chain may have achieved better results, but it is also possible that the objective function for $\mathcal{O}_{det}$ is convex enough that direct ascent is highly efficient.

For the knot based objective functions, we consider the best triangulations achieved overall from both techniques, identifying which edge within the triangle has the most complex knot (as defined by the objective function).

For $\mathcal{O}_{deg}$ the edge that had the greatest degree had a degree of $1,626$. Additionally, it had a fundamental group that was not of the form $\langle a, b \mid a^p b^{-q} \rangle$ and had an Alexander polynomial that could not be decomposed over $\mathbb{Q}$, as such it is not a torus knot, and is likely a prime knot. It is likely a satellite knot ***Jonathan - Can we check this?***. In total for this triangulation, 3 of the edges where unknots.

For $\mathcal{O}_{det}$ the edge that had the greatest determinant had a determinant of 57. The Alexander polynomial factors to $(a^4-a^3+a^2-a+1)(a^{20}-a^{15}+a^{10}-a^5+1)$ and the fundamental group was $\langle ab \mid a^{12}ba^{-13}b \rangle$. This is likely a (5,2)-cable of the (5,2)-torus knot. In total for this triangulation, 3 of the edges where unknots.

For $\mathcal{O}_{norm}$ the edge that had the greatest norm had a norm of 359. The Alexander polynomial had degree 724 and fundamental group with three generators. The Alexander polynomial has alternating $+1, -1$ coefficients suggesting a torus knot (the fundamental group may not be simplified) In total for this triangulation, 3 of the edges where unknots.

For $\mathcal{O}_{gen}$ the edge that had the greatest number of generators had a 51 generators, but after simplification only had 3. The Alexander polynomial had degree 162. The Alexander polynomial has alternating $+1, -1$ coefficients suggesting a torus knot (the fundamental group may not be fully simplified) In total for this triangulation, 4 of the edges where unknots.

For $\mathcal{O}_{var}$ the best triangulation had edges primarily with degree $1, 2$ or 3 and with one edge of degree 131. For this triangulation, all edges where unknotted.

Objective Function	Direct Ascent			Simulated Annealing		
	Score	Total Percentile	Tail Percentile	Score	Total Percentile	Tail Percentile
$\mathcal{O}_{deg}$	152.26	$7.19 \cdot 10^{-05}$	$1.21 \cdot 10^{-05}$	433.42	$2.12 \cdot 10^{-05}$	$2.66 \cdot 10^{-06}$
$\mathcal{O}_{det}$	27.19	$1.38 \cdot 10^{-13}$	$2.62 \cdot 10^{-13}$	13.84	$5.86 \cdot 10^{-11}$	$1.61 \cdot 10^{-10}$
$\mathcal{O}_{norm}$	29.06	$7.16 \cdot 10^{-12}$	$3.29 \cdot 10^{-06}$	152.26	$2.38 \cdot 10^{-17}$	$1.23 \cdot 10^{-08}$
$\mathcal{O}_{gen}$	1.94	$1.58 \cdot 10^{-06}$	$2.32 \cdot 10^{-05}$	2.65	$2.93 \cdot 10^{-09}$	$1.14 \cdot 10^{-07}$
$\mathcal{O}_{var}$	412.54	$3.06 \cdot 10^{-15}$		522.93	$7.52 \cdot 10^{-18}$	

TABLE 2. Comparison of maximum achieved objective function for both classical sampling techniques. Percentiles of each score based on bulk and tail fitted power law distributions.

	Best Triangulation (`IsoSig`)
$\mathcal{O}_{deg}$	`EMzMzMzMzMvAAMzMzMzMPabcdefghijklmnqpqrstuvwxyzABCDDjxxxxxxxxxxxxxlfxfxxxxxxxxxxxj`
$\mathcal{O}_{det}$	`ELLMMzMzMzMzMzMzMzMzQbeddfghijklmnopqrstuvwxyzABCDDDxxrfaxxxxxxxxxxxxxxxxxxxxxxxxxvc`
$\mathcal{O}_{norm}$	`EMzMzMzMzMvAAMzMzMzMPabcdefghijklmnqpqrstuvwxyzABCDDjxxxxxxxxxxxxxlfxfxxxxxxxxxxxj`
$\mathcal{O}_{gen}$	`ELwvMvQPLMzAPLLAPMzPQcbehgklimllpnqptrutwvzxyABDCBDDaacmvlpvlooashoqkmbuoageahamkcg`
$\mathcal{O}_{var}$	`EMzMzMzMzMzMvAAMzMzMPabcdefghijklmnopqsttuvwxyzABCDDvaaaaaaaaaaaaaaaaciqnaaaaaaaaav`

TABLE 3. Best Triangulation Direct Ascent

	Best Triangulation (`IsoSig`)
$\mathcal{O}_{deg}$	`ELvLLzvMvzMQAMLMQzQMQbdgfjpmtosmvruqptvwuzyxBCzACBDDafgfjxaovwrdsifjouitrwblqjfhroj`
$\mathcal{O}_{det}$	`ELLvMLAAzPwMzAPzvAMQQadeihhhljnplnqpturttxBBCADBAADDbabagjgxbadjbhohagkfauahoxrccjo`
$\mathcal{O}_{norm}$	`EvLPzvvzLAvzLPPQQMQQQdcdgklqsvoAptytAzBxxzCwyBBDzADDonnapdeaacqowdbauqadasckkjabvhd`
$\mathcal{O}_{gen}$	`ELMvLAAvMAQzLwPLAMMMPcbdgfiihnmloopnqtsvtxwyzyABACDDaaaavlsvaccfbxsaqranmglanxxnxxr`
$\mathcal{O}_{var}$	`EMLLLvzPPzwPLMQzAPzAQabddglnjokkqnouqswstuvwzyzBCDCDgagbaaagacvaggaccavgfbcafcoaaco`

TABLE 4. Best Triangulation Simulated Annealing

4.5. Baseline Transformer Efficiency on Isomorphism Signatures

A dataset of triangulations of S^3 is collected for a range of different sizes ($6-$tetrahedra to $12-$tetrahedra). These either come from the census of triangulations [**?**], or from MCMC. A separate transformer model is trained for each number of tetrahedra. Each transformer has the exact same parameters: an embedding dimension of 64, 6 layers, 8 heads, a dropout rate of 0.1, and a learning rate of 0.0005. Each model was trained from the same starting key for $50,000$ iterations, where an iteration is a single batch of size 32. Each batch was a randomised sample from the training data. A fixed, random, test set of $1,000$ was used for validation. These parameters were chosen as a computationally reasonable starting point for training, though there is significant research to suggest that larger models will perform better, especially given the essentially unlimited training data that we have [**?**].

Figure 8 shows a log-log plot of the test loss over the number of iterations. It is evident that the curves are still decreasing, and at $50,000$ iterations there isn't significant enough curvature to infer when the training would plateau, this suggests that further training time would yield better results. Regardless of the small architecture and low training time, the data does exhibit promising results.

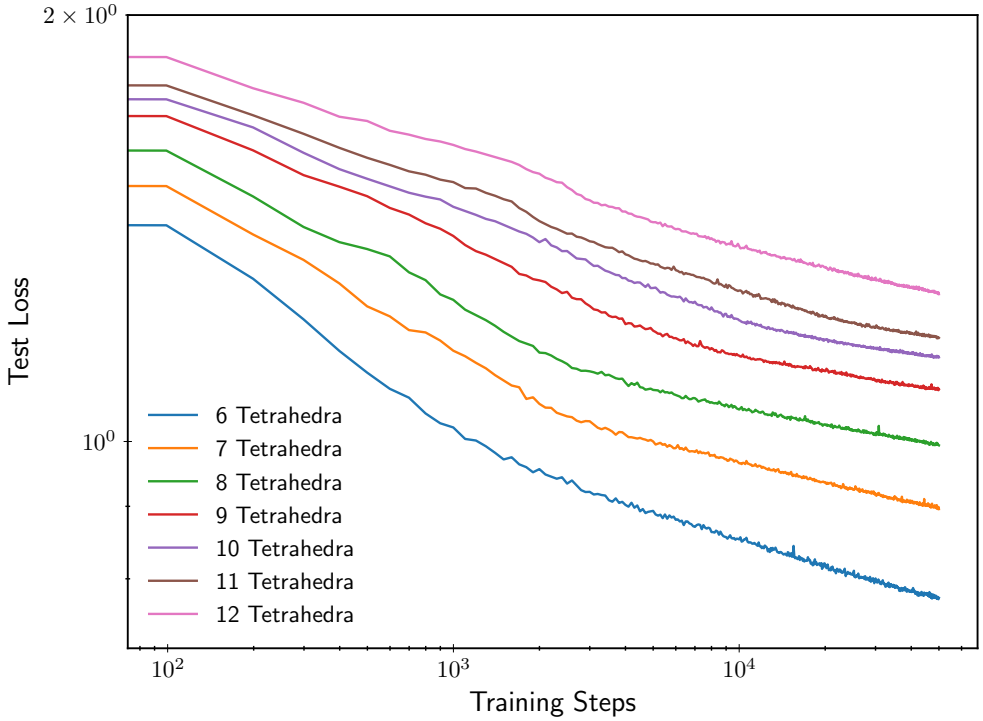


FIGURE 8. Caption

Figure 9 and table 5 show the generation efficiency for the transformers trained on different numbers of tetrahedra. "Loads" indicates if the whether the generated string is formatted correctly to be loaded into regina. "Valid" checks if the string represents a topologically valid triangulation,

i.e. doesn't have any singularities etc. in it. "Closed" checks if the manifold is closed (compact and has no boundary). "Sphere" checks if the manifold represents a sphere, i.e. if its fundamental group is trivial. In general, of the manifolds that load, almost all of them are valid. Also, of the manifolds that are closed about 70% are spheres - though there is not enough data to suggest if this holds true in general. However, the fraction of valid manifolds that are closed does show evidence that it is decreasing as the number of tetrahedra increases.

# Tetrahedra	Loads	Valid	Closed	Sphere
6	0.195	0.195	0.115	0.086
7	0.08	0.08	0.039	0.0235
8	0.043	0.0425	0.02	0.0115
9	0.0185	0.0185	0.004	0.004
10	0.008	0.008	0.0025	0.001
11	0.0075	0.0075	< 0.001	< 0.001
12	0.001	< 0.001	< 0.001	< 0.001

TABLE 5. Generation efficiency at different number of tetrahedra.

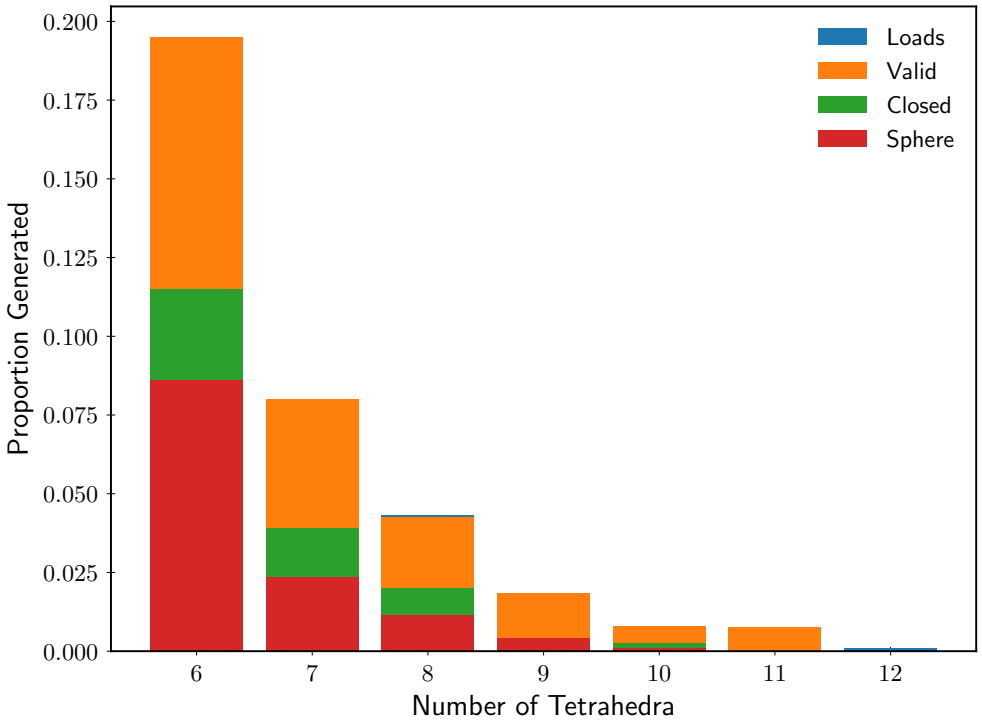


FIGURE 9. Caption

CHAPTER 5

Discussion and Future Work

5.1. Guided Search

During the development of the optimisation techniques, there was interest in understanding if guided search could be used to discover specific, special types of triangulations. In particular, we were interested to see if we could use guided search to discover a triangulation where all the edges were knotted. Burton had used targeted search to find an example where this was true. The concept was that if we find a triangulation which exhibits particularly high "knottedness" in general, it may be more likely to have all of its edges knotted. However, after running optimisation on a range of different objective functions, we found that we could achieve much higher values for the objective functions than what was actually obtained by the triangulation discovered by Burton. That is, the objective functions did not discriminate enough to be used to locate such a triangulation. This is not a failure in the optimisation techniques, but rather in the definition of the objective function. In general, if the discussed optimisation techniques are to be used for searching for specific triangulations, it needs to be done on some objective function that is known to be optimised by a hypothetical solution. For example - the number of edges unknotted is known to be minimised by a triangulation of the desired form (tautologically), unfortunately it is too flat to actually serve any value for searching (tending to "hover" around 2 quite easily).

5.2. Reinforcement Learning

We have successfully trained transformers to be able to generate triangulations, though the efficiency isn't particularly high. Before deciding to do reinforcement learning on the objective function, it may be beneficial to improve the rate at which the model is generating valid spheres. Apart from increasing the amount of training time, one possible avenue would be to do reinforcement learning with the objective being whether or not the triangulation is a sphere (or to start simply if its valid). In order to apply reinforcement learning, it is generally required that the number of positive samples per batch is ≥ 1. Considering our batch size is 32, this would mean the efficiency needs to be above 0.03. This is achieved for $6-$tetrahedra directly for spheres, and for $6, 7, 8-$tetrahedra for the initial task of being valid. The batch size could certainly be increased though this would require more compute power. As such, reinforcement learning is a valid avenue immediately for triangulations of size $6, 7$ and 8, and would likely apply to larger triangulations after training with gradient descent for longer and with a larger model.

5.3. Alternate Isomorphism Signatures

It is a well empirically observed result that increasing the embedding dimension increases performance for transformers. This can be explained with two factors. Firstly, in general more parameters of any form gives the model more freedom and nuance allowing it to fit to the data better. Secondly, and more pertinent, the embedding layer is the component of the model that learns to express what each letter "means". With the current isomorphism signatures, they are simply representations of numbers, so specific letters don't contain any geometric meaningful information, the model has to "waste effort" learning how to decode the isomorphism signatures, which is a relatively sophisticated encoding structure. A different encoding strategy could be used to represent the triangulations where each token has a specific and unique meaning. One possibility is "dehydrated isomorphism signatures". These have a potential advantage because each pair of letters encodes a unique piece of geometric information, and the position encodes the gluing information. This is far more subtile for the specific task that transformers were developed to handle and using such a signature could lead to faster training times, and higher accuracy for the same sized model. Additionally, there could be potential interest in the embedding vectors learned themselves.