ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

GIRG model for realistically generating social network graphs

Master Thesis

Benjamin Dayan

August 30, 2023

Advisors: Prof. Dr. Angelika Steger, Prof. Dr. Johannes Lengler, Marc Kauffman, Ulysse Schaller

Department of Computer Science, ETH Zürich

Abstract

Many real-world graphs, in particular social networks, exhibit clustering and are hypothesized to possess an underlying geometrical generative structure. The Geometric Inhomogeneous Random Graphs (GIRG) model is one proposed generative geometric model. This thesis demonstrates the proficiency of assorted GIRG variants in generating graphs that resemble a dataset of real social network graphs across a spectrum of high-level graph features. We then investigate practical methods of fitting GIRG node geometric location parameters to a given input graph. Using these, we show that low-dimensional GIRGs can successfully replicate our social network graphs on a node and edge level. Throughout this thesis, we aim to offer guidance on generating and fitting different types of GIRGs, as well as an intuitive understanding of the GIRG model.

Contents

Chapter 1

Introduction

1.1 Thesis Introduction

Generative models for real-world graphs Graphs are a useful model for describing networks of interconnected entities as an abstraction of real-life systems. Real-world graph data are abundant, and many graphs exhibit similar structural characteristics. Several **graph-generating models** have been proposed, each justified by its reasonable formative mechanisms which create sets of graphs similar to those observed in the wild. Studying these models theoretically provides insights into real networks and is useful for creating synthetic graphs for testing purposes. However, these models are only practically useful conditioned on their ability to well match real graph data, which is a challenge to quantify.

Geometric Inhomogeneous Random Graphs (GIRGs) are one such generative model, first proposed in [Bringmann et al., 2016]. They are a generalisation of **Hyperbolic Random Graphs (HRGs)**, put forward in [Krioukov et al., 2010] and quickly shown to have good use in modelling the internet routing graph, in [Boguná et al., 2010]. GIRGs allow significant flexibility in the geometric component of their generative structure, thereby enabling better modeling of a broader range of real graphs; HRGs are essentially a type of 1-dimensional GIRG. GIRGs' definition encapsulates the intuitive notion present in many graphs that nodes are more likely to be connected if they are "close together", often observed as **clustering**. Furthermore, GIRGs exhibit inhomogeneity between nodes, resulting in a power law degree distribution, a characteristic observed in many real graphs.

Matching GIRGs to Facebook social network graphs There has been much good theoretical research in analysing the properties of GIRGs, however not enough empirical work on assessing the quality of fit of the GIRG model to real graphs. This thesis aims to evaluate the ability of GIRGs to simulate a dataset of 104 social network graphs obtained from the Network Repository ([Rossi and Ahmed, 2015], `https://networkrepository.com/`). These graphs represent Facebook social networks from various towns in the US, thus exhibiting structural similarities, and ranging from $n = 762$ to $n = 35,111$ nodes. In chapter 3, we assess the mostly unmodified GIRG prior distribution of graphs based on their high-level similarities

to the real graphs. In chapter 5, we perform a more intensive posterior style fit of GIRG to replicate real graphs at the node/edge level.

High-level GIRG realism using Bläsius framework The main novel research contribution of this thesis is in chapter 3. We assess the GIRG prior's realism by using a high-level real/generated graph dataset comparison framework presented in [Bläsius et al., 2018a]. This framework was proposed to better ground applicability of theoretical generative models to real-world graph data, by asssessing similarity between real and generated graphs in different high-level graph statistics. It gives full choice on which statistics to compare, and performance metrics for each (or arbitrary combination of) statistic. The models originally assessed in [Bläsius et al., 2018a] included HRGs and others, but not GIRGs. Our results provide evidence that GIRGs are more realistic candidates than simpler generative models with respect to geometric features such as node closeness centrality, betweenness centrality, local clustering coefficient, and graph effective diameter. We compare the realism of different GIRG variants on these features and find that while max norm torus GIRGs provide a good baseline, performing particularly well on betweenness centrality, cube GIRGs perform better on graph effective diameter, and mixed minimum component distance GIRGs excel in closeness centrality. In most (but not all) cases, we find that a small geometric dimensionality of $d = 1$ already provides near-peak GIRG realism, with little benefit from higher-dimensional GIRGs.

Fitting GIRG node geometric locations The aim of "posterior fitting" is to infer the node locations of a given posited GIRG generative model for an input real graph. Work by [Boguná et al., 2010] and later [García-Pérez et al., 2019], [Bläsius et al., 2018b] and [Bläsius et al., 2021] all show solutions to this problem for HRGs. We apply variants of the first two papers' methods to GIRGs, and despite lack of novelty and questionable implementation on our part, we find some interesting tidbits along the way. In chapter 4, we explore the method of diffusion maps as a computationally fast initial estimate of locations with good internode distances, and as a means of proposing the underlying GIRG geometric dimensionality, instead of a more theoretical method like in [Friedrich et al., 2023]. In chapter 5, we describe methods to further refine node locations using either a markov chain Monte Carlo or maximum-likelihood-based fitting approach. The full fitting procedure exhibits decent performance in replicating our real social networks, and supports the hypothesis that one dimension is mostly sufficient for this dataset.

GIRG variants definitions and sampling To support all objectives, in chapter 2 we cover the main different types of GIRG: torus/cube; max norm, minimum component distance, mixed min/max, and distorged. We also detail practical algorithms for sampling/generating a GIRG.

Our (python) code is available at `https://github.com/BenjiDayan/GIRGs`, however code documentation and quality is not guaranteed.

Chapter 2

GIRGs - Variants and Generation

Contents

2.1 GIRGs definition and key features

As defined in [Bringmann et al., 2019], the GIRG model is a random graph model characterized by the edge connection probabilities given by the formula:

$$p_{uv} = \Theta\left(\min\left\{1, \left[\frac{w_u w_v / W}{\|x_u - x_v\|^d}\right]^\alpha\right\}\right); \quad \rho_{uv} = \frac{w_u w_v / W}{\|x_u - x_v\|^d} \tag{2.1}$$

$(w_u)_{u\in V}$ are node-specific weights in $\mathbb{R}^+$, and $W = \sum_{u\in V} w_u$. Nodes are given also a location $x_u \in \chi$ in a particular geometric space χ, often taken to be the d-dimensional torus [1] $\mathbb{T}^n$, or the d-dimensional unit cube $[0,1]^d$. The $\min(1,\cdot)$ ensures

[1] The d-dimensional torus also resembles $[0,1]^d$, but opposite faces of the cube are identified. Picture Pac-Man, or donuts. Just not Pac-Man eating a donut...

that probabilites $p_{uv} \leq 1$. The outer $\Theta(...)$ specifies that we allow some wiggle room, that $c_1(...) \leq p_{uv} \leq c_2(...)$ across all potential edges, for some constants $0 \leq c_1 \leq c_2 < 1$. For theoretical statements about the asymptotic behaviour of GIRGs as $n \to \infty$, these constants must be fixed across all n.

We will dive more into the significance and intuition behind eq. (2.1) edge probabilities in the following subsections.

2.1.1 Geometry

The edge probability p_{uv} incorporates a geometric factor from the distance $r_{uv} = r(\boldsymbol{x}_u, \boldsymbol{x}_v)$, which inversely scales the probabilities so that nearby points (small r_{uv}) are more likely to have an edge than those further apart. r_{uv} is raised to the dth power to ensure consistency in the number of edges across different numbers of dimension d. The influence of node proximity on p_{uv} in GIRGs facilitates the emergence of *clustering*, a common phenomenon in many real-world graphs, where subgroups of nodes might have more internal edges than would be expected by chance.

The geometric space χ, along with the distance function, often written $r(\boldsymbol{x}_u, \boldsymbol{x}_v) = \|x_u - x_v\|$ (which is not necessarily a norm, or even a metric), can vary a great deal without affecting key properties of GIRGs. A higher dimensional space χ can have greater expressive power, with more complicated patterns emerging from the geometry. Node locations are assumed to be distributed uniformly in the space, $\boldsymbol{x}_u \sim U(\chi)$, which is fine as a prior, although in real-world geometries this is often not very accurate.

Using the *max norm* as the distance function, i.e., $\|\boldsymbol{x}\| = \|\boldsymbol{x}\|_\infty = \max_i |x_i|$, is convenient for proofs, but can be replaced by euclidean or other norms - by the equivalence of norms there is not much difference. Note that on the torus, the max norm is not strictly a norm, rather we abuse notation to express:

$$||\boldsymbol{x} - \boldsymbol{y}||_\infty = \max_{i \in [d]} |x_i - y_i|_{\mathbb{T}} \qquad \text{(in a torus } \mathbb{T}^d\text{)} \tag{2.2}$$

$$|a - b|_{\mathbb{T}} = \min(|a - b|, 1 - |a - b|) \tag{2.3}$$

The *minimum component distance (MCD)*, defined as $\|x\|_{\text{mcd}} = \min_i |x_i|$, does not even satisfy the triangle inequality, yet it preserves most properties of GIRGs. This distance function is reasonable if we desire two nodes to have a higher edge probability if they are close in at least one dimension, rather than in every dimension.

$\alpha \in (1, \infty]$ is a parameter that impacts the affect of geometry on edge probabilites. We differentiate between *short edges* where $r_{uv} < (w_u w_v / W)^{1/d}$ and $p_{uv} = \Theta(1)$ (for these, the $\min(1, \cdot)$ is applied), and *long edges*, where $r_{uv} > (w_u w_v / W)^{1/d}$, and p_{uv} diminishes to zero as the distance increases. Larger values of α accelerate this decay in long edge probability, thus more severely limiting the number of long-distance edges. $\alpha = \infty$ imposes a strict cutoff, setting $p_{uv} = 0$ for long edges.

2.1.2 Similarity to Chung-Lu

A key property of the GIRG model is its nature as a geometric variant of the *Chung-Lu* random graph model. In a Chung-Lu generated graph, edge probabilities

between nodes for a given node weight sequence $(w_u)_{u\in V}$ are decreed as

$$p_{uv} = \Theta\left(\min\left\{1, \frac{w_u w_v}{W}\right\}\right) \tag{2.4}$$

In a GIRG, geometry affects edge probabilities, yet when considering any two nodes u, v, marginalising over the uniformly distributed node locations (which affects edge prob p_{uv} by the distance r_{uv}) yields

$$E_{\boldsymbol{x}_v}[p_{uv}|\boldsymbol{x}_u, w_u, w_v] = \Theta\left(\min\left\{1, \frac{w_u w_v}{W}\right\}\right) \tag{2.5}$$

which mirrors the Chung-Lu model and disregards node locations $(\boldsymbol{x}_u)_{u\in V}$.

Therefore, properties and results pertaining to graphs generated by the Chung-Lu model are directly applicable to GIRGs, such as the fact that $E[d_u] = \Theta(w_u)$: the expected degree of a node u with weight w_u is proportional to w_u. This makes sense, as the min in the edge probability only rarely applies, so for most potential edges out of a node u, the probabilities scale linearly with w_u. Due to the normalising factor of $W = \sum_{u\in V} w_u$, there is no dependence of the expected degree of u on n. As $n \to \infty$, although u will have more possible other nodes to connect with, this increase is counterbalanced by the declining probability of connecting to any one of them. Hence we will often think of a node's weight and its degree as equivalent.

eq. (2.5) might appear counterintuitive, given the presence of the $(w_u w_v/W)^\alpha$ term in the edge probabilities in eq. (2.1). However, as previously discussed, α only affects the decay rate of edge probabilities for long edges, which constitute only a constant fraction of the total edge probability, provided $\alpha > 1$. If α were smaller, it would blow up the degree d_u far beyond w_u, as the abundance of potential long edges would result in too many being realised as actual edges, so would dominate the number of short edges.

2.1.3 Power Law Degree Sequence

The weight sequence $(w_u)_{u\in V}$ of nodes in a GIRG and hence node degrees, are usually assumed to be generated from a power law distribution with an exponent $\tau \in (2,3)$. This can also be loosened to just include the tail (larger) weights, for instance just the top 10% largest weight nodes. This allows the GIRG to match the common real network phenomenon of having a degree distribution with a tail that looks like a power law.

$$w_u \sim \text{powerlaw}(\tau) \qquad \text{exact power law distribution} \tag{2.6}$$

$$p(w_u = w) \propto w^{-\tau} \text{ for } w \in [w_{\min}, \infty] \qquad \text{pdf (default } w_{\min} = 1) \tag{2.7}$$

$$p(w_u \geq w) \propto w^{1-\tau} \qquad \text{cdf} \tag{2.8}$$

The power law distribution is heavy tailed (heavier than exponentially decaying tails). $\tau > 2$ is important to ensure that $E[\mathrm{w}] = \Theta(1)$, which means that although we may have some very large w_i in a sequence $w_1, ..., w_n$, still there is an equal balance of total weight between smaller valued w_i's and larger weights. This produces a phenomenon known as the "Pareto principle / 80-20 rule", a common example being the top 20% richest individuals collectively owning roughly 80% of total private wealth. That's certainly a heavier tail than an ideally fair society, but it also doesn't completely skew the mean wealth of an individual more than some multiple of the median wealth.

2.1.4 Volume generalisation

The volume formulation of a GIRG presented in [Bringmann et al., 2019] provides a consistent way to handle different geometries, modifying ρ_{uv} in eq. (2.1) to be

$$\rho_{uv} = \frac{w_u w_v / W}{Vol(r_{uv})} \tag{2.9}$$

Here $Vol(r_{uv}) = Vol(B_{r_{uv}})$ is the volume of the ball of radius r using the distance function $r(x_u, x_v) = r_{uv}$, which should be symmetric. For example with the max norm, $Vol(r_{uv}) = (2r_{uv})^d$ as a cube with side-length $2r_{uv}$.

Having volume in the edge probability formula is actually a generalisation of taking r_{uv} to the dth power - the point being to make $p(u \sim v | x_u, w_u, w_v) = E_r[p_{uv}(r)] = \Theta(w_u w_v / n)$, regardless of dimension. We'll actually derive in section 3.2.5 a precise formula for expected edge probabilities / node degrees that holds across all different types of volume formulation torus GIRGs. This will however need an exact GIRG edge probability formula, without the outer $\Theta(\cdot)$, which we present in section 2.2.

2.2 Blasius C++ GIRG generation algorithm

[Bläsius et al., 2022] provides a C++ implementation of an efficient algorithm for sampling a *max torus GIRG* - with the max norm as distance function and torus geometric space. Their GIRG formulation uses a toroidal geometry $\chi = \mathbb{T}^d$, with edge probabilities

$$p_{uv} = \min\left\{1, c\left(\frac{w_u w_v / W}{\|x_u - x_v\|_\infty^d}\right)^\alpha\right\} \tag{2.10}$$

i.e. no outer Θ which gave us more flexibility in the general GIRG definition - just one fixed inner constant c in order to be concrete. Note that this still falls under the wider Θ GIRG definition as p_{uv} always lies in the interval

$$c \ \min\{1, \rho_{uv}^\alpha\} \leq p_{uv} \leq \min\{1, \rho_{uv}^\alpha\} \tag{2.11}$$

if $c \leq 1$, and with the upper and lower bounds swapped for $c > 1$.

They implement the algorithm proposed by [Bringmann et al., 2019], which has expected runtime $O(n)$ i.e. linear in the number of nodes, for a GIRG with a power law generated weight sequence. This is optimal because such a GIRG has expected number of edges $\Theta(n)$ whp, due to the fact that nodes have finite expected weight $E[w_u] = \Theta(1)$, and expected degree proportional to their weight. I.e. just to write down / store the set of edges $E = \{(u, v) | u \sim v\}$ we need $O(n)$ time/space.

2.2.1 Naive GIRG generation algorithm with Python

Unfortunately this GIRG sampling algorithm scales exponentially in dimension d, as it involves breaking the Torus space down hierarchically into smaller cubes and considering edges between neighbouring cubes. We struggled to use it in experiments for $d \geq 4$. Hence we also implemented our own GIRG generation code in python for ease of modification to deal with different types of GIRG and to cope with higher dimensions d. Our algorithm is the most basic $O(n^2)$ checking of every

single node pair (u, v), and also has $O(n^2)$ memory requirement, and doesn't scale in d beyond a linear factor inherent in operating with d-dimensional points. The basic steps are:

1. sample $O(n)$ node weights w_u from a power law distribution
2. sample $O(n)$ node locations $\boldsymbol{x}_u$ from a uniform distribution on the torus
3. compute the $O(n^2)$ pairwise node distances and hence edge probabilities
4. for each of the $O(n^2)$ potential edges, sample a Bernoulli random variable with the edge probability as its parameter, to determine whether the edge exists

2.3 Cube GIRGs

Cube GIRGs are an alternative formulation to torus GIRGs. Instead of the d-dimensional torus $\chi = \mathbb{T}^d$, we have the d-dimensional cube $\chi = [0,1]^d$. Thus, the tradeoff is to lose the symmetry and simplicity (useful for theoretical proofs) of the torus for the ability to hopefully generate more realistic graphs, as few situations in the real world are toroidal i.e. with features that wrap around in the way that a torus does.

For example in our socfb social network graphs, likely geometric features for each individual could be home address, age, athletic proclivity, political leaning etc. In the environ of a town/city, home address is a non-torioidal 2d vector[2]. Age and athleticism are clearly both non-toroidal (and hence cube-like) quantities. Political leaning could be, e.g., a 2d axis of economic left/right and authoritarian/libertarian. This might bear a toroidal-esque characterisation whereby two individuals on far opposite extremes of an axis share some common ground, e.g., a hardcore communist and a fascist might both share values such as railing against the status quo or enjoying political rallies. No model is perfect; this could potentially be captured as an extra non-toroidal feature dimension placing individuals on a scale from mainstream to unconventional.

2.3.1 Cube GIRG Formulation

The neat correspondence between volume and distance unfortunately breaks down when translating from torus GIRGs to cube GIRGs, due to the loss of spatial symmetry. With the max norm,

$$\text{in the torus} \qquad Vol(r_{uv}) = (2\,\|x_u - x_v\|_\infty)^d \tag{2.12}$$

$$\text{in the cube} \qquad B_{r_{uv}}(u) = [0,1]^d \cap \bigtimes_{i=1}^{d} \left((\boldsymbol{x}_u)_i - r_{uv}, (\boldsymbol{x}_u)_i + r_{uv}\right) \tag{2.13}$$

i.e. the ball's volume is cutoff when exceeding the edges of the cube $[0,1]^d$ rather than wrapping like in the torus. Hence oftentimes $Vol(B_{r_{uv}}(u)) \neq Vol(B_{r_{uv}}(v))$.

[2] Only on the whole planet scale would home address look more torus like - strictly speaking a 3d sphere, or perhaps a cylinder if you assume that nobody lives near to the North/South Pole. [García-Pérez et al., 2019] embeds a worldwide aiport network graph as a HRG, i.e. a 1d torus GIRG, whose inferred node locations do roughly match with longitude. [Boguná et al., 2010] embeds the internet graph as a HRG, with definite continental clustering but less clear correspondence to longitude

Distance based formulation We will use a simpler *distance based formulation* of the cube GIRG. This simply uses a cube based distance function $r^{\text{cube}}(\boldsymbol{x}_u, \boldsymbol{x}_v)$ in edge probability formula p_{uv} in eq. (2.1), a distance that cannot short cut by wrapping round the torus. E.g. $r^{\mathbb{T}}(0.1, 0.9) = 0.2$, whereas $r^{\text{cube}}(0.1, 0.9) = 0.8$. This is conceptually easy to understand, and allows for a neat coupling with the previous torus geometry whereby

$$r_{uv}^{\text{cube}} \geq r_{uv}^{\mathbb{T}} \quad \text{for all pairs of nodes } u, v \tag{2.14}$$
$$r_{uv}^{\text{cube}} = r_{uv}^{\mathbb{T}} \quad \text{for most pairs} \tag{2.15}$$
$$p_{uv}^{\text{cube}} \leq p_{uv}^{\mathbb{T}} \quad \text{consequent } \textbf{stochastic domination} \tag{2.16}$$

Hence a cube GIRG is strictly sparser than its torus GIRG counterpart.

Volume based formulation is a more complicated alterative that we do not employ. This could look something like replacing $Vol(r_{uv}) \mapsto \sqrt{Vol(B_{r_{uv}}(u))Vol(B_{r_{uv}}(v))}$. Intuitively in the social network analogy, this is like saying that all people, no matter how extreme their geometric location (near the edge of the cube), have the same desire to make friends (modulo their inhomogeneous weights as that is the literal introversion / extroversion factor in the GIRG model). If instead of having undirected edges $u \sim v$ we had directed edges $u \to v$, we could even have a model where $p(u \to v) \propto \sqrt{Vol(B_{r_{uv}}(u))}$, such that u would in expectation have the same number of outgoing edges whatever its location $\boldsymbol{x}_u$. Unfortunately desire to make friends doesn't equate to actual friends, as they have to want you back. So if a node u is near the edge of the cube, while it would send out the normal volume based amount of friendship invitations ($u \to v$), as most of these go to nodes v nearer the centre of the cube, fewer will reciprocate ($v \to u$). In the directed torus geometry analogy, due to symmetry, every $p(u \to v) = p(v \to u)$. This means more pairs of $u \leftrightarrow v$, and fewer pairs of "unrequited friendships" i.e. $u \to v; v \not\to u$. Hence a volume based cube GIRG in the undirected setting would still have higher average degree than the distance based version, but lower than the torus.

2.3.2 Cube GIRG generation - coupling algorithm

We use the distance based cube GIRG formulation, which permits a simple coupling to a torus GIRG. This motivates algorithm 1 which generates a cube GIRG by appropriately randomly deleting edges from a pre-generated torus GIRG . All we need is any black box torus GIRG generation algorithm that in addition to the graph edge list, also yields the sequence of node weights and locations. algorithm 1 runs in $O(|E|)$ time, additional on top of the torus GIRG generation runtime. As GIRGs have $|E| = \Theta(n)$ whp, this means whp $O(n)$ additional time.

Since $p_{uv}^{\text{cube}} \leq p_{uv}^{\mathbb{T}}$, we can couple the events $u \sim v$ in the cube GIRG as a subset of the events $u \sim v$ in the torus GIRG. Thus we just have to inspect each of the $\Theta(n)$ edges in the pre-generated torus GIRG, and flip a Bernoulli$(p_{uv}^{\text{cube}} / p_{uv}^{\mathbb{T}})$ coin for each to decide whether to keep / delete the edge for the cube GIRG.

Algorithm 1 Generate cube GIRG from torus GIRG via coupling

Require: n, d, c, τ, α
  $(G, \{x_u\}_{u\in V}, \{w_u\}_{u\in V}) \leftarrow$ torus-GIRG(n, d, c, τ, α)
  for $(u, v) \in E(G)$ **do**
    $p_{uv}^{\mathbb{T}} = \min\{1, c\left(\frac{w_u w_v/W}{(r_{uv}^{\mathbb{T}})^d}\right)^\alpha\}$
    $p_{uv}^{\text{cube}} = \min\{1, c\left(\frac{w_u w_v/W}{(r_{uv}^{\text{cube}})^d}\right)^\alpha\}$
    $p \leftarrow U[0,1]$
    if $p > \frac{p_{uv}^{\text{cube}}}{p_{uv}^{\mathbb{T}}}$ **then**
      delete edge (u, v) from G
    end if
  end for
  return G

2.4 Minimum Component Distance (MCD) GIRGs

In section 2.1 we introduced the *minimum component distance (MCD)* as an alternative distance function to the max / euclidean norms,

$$\|\boldsymbol{x} - \boldsymbol{y}\|_{\text{mcd}} = \min_i |x_i - y_i| \tag{2.17}$$

While the max norm is commonly used in e.g. [Bringmann et al., 2019] as it is handy for proofs, the MCD can make more sense in some settings. For instance in social networks, the max norm is like stipulating that people only make friends with others that "tick all the boxes", i.e. are similar in every dimension (this sounds more similar to the criteria in finding a life partner). The MCD is saying that you make friends with people who are similar in at least one dimension. E.g. an individual might play with a neighbourhood football group, or sing in a local choir - each a social circle formed from one shared hobby.

The MCD can be mixed with the max norm in an "and/or" fashion, e.g. on points in 5d one grouping is

$$\|\boldsymbol{x} - \boldsymbol{y}\|_{\text{mix}} = \min\left(||\boldsymbol{x}_1 - \boldsymbol{y}_1||_\infty, ||\boldsymbol{x}_{[2,3]} - \boldsymbol{y}_{[2,3]}||_\infty, ||\boldsymbol{x}_{[4,5]} - \boldsymbol{y}_{[4,5]}||_\infty\right) \tag{2.18}$$

This parses as points $\boldsymbol{x}, \boldsymbol{y}$ being "close" if they are close in the 1st dim, or the 2nd and 3rd dim, or the 4th and 5th dim. We call this a *mixed min/max GIRG*.

2.4.1 Volume Formula for MCD, and Mixed Norms

The formula for $Vol(r)$ differs according to the distance function $r(\boldsymbol{x}, \boldsymbol{y})$ and geometric space χ that is in use. For the max norm in d dimensions we saw that $Vol(r) = (2r)^d$, as the ball B_r in this norm is a cube with side-length $2r$. For the euclidean norm this becomes $Vol(r) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d$ as the ball B_r is in this case really "ball shaped". E.g. in $d = 1, 2, 3$ dimensions the ball volumes are respectively $2r$; πr^2; $\frac{4}{3}\pi r^3$.

For the MCD, we can derive a volume formula by viewing ball volumes through the lens of probabilistic events. Consider the point set $B_r(\boldsymbol{x})$, and a location sampled

uniformly randomly from the torus $\boldsymbol{y} \sim U(\mathbb{T}^d)$. Let event A_i be that $|y_i - x_i| < r$. Then $Vol(r) = P(\bigcup_{i=1}^d A_i)$, as the whole torus has unit volume. Since events A_i are mutually independent by orthogonality of the individual dimensions, we have that

$$Vol(r) = P\left(\bigcup_{i=1}^d A_i\right) = 1 - P\left(\bigcap_{i=1}^d \overline{A}_i\right) \tag{2.19}$$

$$= 1 - \prod_{i=1}^d P(\overline{A}_i) = 1 - (1-2r)^d \tag{2.20}$$

This can be extended to mixed min/max GIRGs with $(I_i)_{i=1}^k$ a partition of $\{1, ..., d\}$

$$r(\boldsymbol{x}, \boldsymbol{y}) = \min\left(\|\boldsymbol{x}_{I_1} - \boldsymbol{y}_{I_1}\|_\infty, ..., \|\boldsymbol{x}_{I_k} - \boldsymbol{y}_{I_k}\|_\infty\right) \tag{2.21}$$

$$\text{then} \quad Vol(r) = 1 - \left[\prod_{i=1}^k 1 - (2r)^{|I_i|}\right] \tag{2.22}$$

2.4.2 MCD GIRG speedier combination of 1d GIRGs generating algorithm

We introduced in section 2.2 the max torus GIRG generation algorithm implemented in [Bläsius et al., 2022], which has a runtime of $O(n)$ for fixed dimension d. Unfortunately this scales exponentially in d, which hinders our experiments when generating large GIRGs (e.g. $n \geq 10,000$ with $d \geq 4$). We propose a faster algorithm that generates an MCD GIRG by first generating d 1d GIRGs each in $O(n)$ time, and combining them together into one graph in total $O(dn)$ time. This significant speedup could bias a practitioner to using MCD GIRGs when high dimensions are required.

To give an intuition if we want to generate an MCD GIRG of dim $d = 3$, it actually looks quite like a union of 3 one dimensional GIRGs G_1, G_2, G_3, each sharing the same node weight sequence, but with independently 1d distributed node locations. Critically, the majority of edges in the resultant 3d MCD GIRG G occur between nodes which happen to be quite close in just one dimension, and not in the other dimensions. This means that the collective number of edges in the three 1d GIRGs is not significantly more than in the end graph, so the complexity of generating these three and handling all their edges won't blow up our runtime too much.

Converting intuition to an algorithm If $\alpha = \infty$ the "union of 1d GIRGs" statement is almost true. For a node pair (u, v), let r_1, r_2, r_3 be the distances in the 3 1d GIRGs and p_1, p_2, p_3 the corresponding edge probabilities. WLOG letting $r_1 \leq r_2 \leq r_3$, then

$$\alpha = \infty \implies p_i \in \{0, 1\} \text{ which means that } u \overset{3}{\sim} v \implies u \overset{2}{\sim} v \implies u \overset{1}{\sim} v \tag{2.23}$$

This means we could directly take the union over all edges in G_1, G_2, G_3 to produce our 3d MCD GIRG G. In the $\alpha < \infty$ case, to decide if the edge (u, v) is present in the final graph G, we rather must additionally check whether the 1d GIRG with the smallest r_i has an edge. E.g. it could be that $p_1 = 0.5, p_2 = 0.3, p_3 = 0.01$, and that $u \sim v$ only in G_2 - in this case we would take $u \not\sim v$ in G as the minimum component is $i = 1$, thus we copy edge/non-edge for (u, v) from G_1.

There's a subtle difference however in that the 3 1d GIRGs actually need to be generated from edge probabilities with a similar formula to those in the resultant MCD GIRG. This means taking [3]

$$p_i = \min\left\{1, c\left(\frac{w_u w_v / W}{1-(1-2r_i)^3}\right)\right\} \tag{2.24}$$

rather than the standard 1d GIRG edge probability which just has $2r_i$ in the denominator. This pleasingly gives us that

$$r_{uv} = \min_j r_j \leq r_i \implies p_i \leq p_{uv} \; \forall i = 1, ..., d \tag{2.25}$$

where p_{uv} and r_{uv} are the edge probability and node pair distance respectively in the final 3d MCD GIRG. This means that the expectated number of edges in each 1d GIRG is fewer than in the resultant MCD GIRG, which is itself whp $O(n)$ - this matches our "union of 1d GIRGs" intuition.

The number of overcounted (and hence needed to be discarded) edges from the 1d GIRGs will therefore exceed the final edge count by at most a factor of d, hence the overall $O(dn)$ runtime of the algorithm. In the standard GIRG model with uniformly distributed point locations we expect $o(n)$ overcounted edges, mostly between nodes u, v where w_u, w_v are both large (like $\Theta(n^{1/2})$). For most nodes of smaller weight $w_u, w_v = \Theta(1)$, overcounted edges require the intersection of two $\Theta(n^{-1})$ probability, independent events, and are hence exceedingly rare. Still in our $d = 3$ example, on the off chance that node locations all lie close to a 1d subspace like $x_1 = x_2 = x_3$, then we would get almost exactly triple counting.

Algorithm 2 Generate MCD GIRG from combination of 1d GIRGs

Require: n, d, c, τ, α
  $(G_i = (V_i, E_i))_{i=1}^d \leftarrow$ 1d-GIRG(n, d, c, τ, α) ▷ edge probs using d-dim MCD GIRG formula
  $V \leftarrow [n]$; $E \leftarrow \{\}$
  for $i \in [d]$ **do**
    for $(u, v) \in E_i$ **do**
      if $i = \arg\min_i r_i$ **then**
        $E \leftarrow E \cup \{(u, v)\}$
      end if
    end for
  end for
  return $G = (V, E)$

2.4.2.1 Extending the algorithm to max GIRGs and mixed GIRGs

Extension to max GIRGs We would like to extend algorithm 2 to generate max norm GIRGs. Unfortunately, although the algorithm is still correct, its runtime blows up due to the fact that each 1d GIRG will now have many more than $\Theta(n)$

[3] This is treacherous as with such an edge probability these "1d GIRGs" might technically no longer be actual GIRGs - however in this section we will still refer to them as such for ease of discussion. Note that having $\Theta(n)$ edges is a key GIRG property, which the "1d GIRGs" in the MCD algorithm still possess; this doesn't hold when applying the algorithm in the max norm setting.

edges. In short, now $r_{uv} = \max_j r_j \geq r_i$, such that $p_i \geq p_{uv}\ \forall i = 1, ..., d$ and each 1d GIRG in expectation has more edges than the resultant max GIRG. This would be ok if there were only a constant factor more, but they actually each have $\Theta(n^{1-1/d})$ times more edges, which means the runtime of the algorithm is now $O(d \cdot n^{2-1/d})$.

We will just give an informal intuition as to why the 1d GIRGs would now have so many more edges. For a node u of fixed weight, consider the neighbours of u of small weight w_v, connected to u by a short edge, in both the resultant d-dim max GIRG, and one of the individual 1d GIRGs. In both GIRGs, these edges constitute whp a constant fraction of all of u's edges. In the resultant d-dim max GIRG, these neighbours are precisely all the low weight nodes inside the box of side length roughly $2r = \left(\frac{\overline{\deg}}{n}\right)^{1/d}$ centered at $\boldsymbol{x}_u$, For a total of $n(2r)^d = \Theta(\overline{deg})$ neighbours. In contrast, in the 1d GIRG corresponding to the ith dimension, the whole strip of nodes whose projection on dim i lie in the range $[(\boldsymbol{x}_u)_i - r,\ (\boldsymbol{x}_u)_i + r]$ will end up as neighbours of u. This gives u a much higher degree in the 1d GIRG, of at least $\widetilde{\deg} = 2nr = \Theta(n^{1-1/d})$ in expectation.

This means that generating a d-dimensionsal max GIRG with our combination of 1d GIRGs algorithm has runtime at least $O(d \cdot n^{2-1/d})$. This would only beat an exponential in d, linear in n, $O(e^{kd}n)$ runtime for $d > \frac{\log n}{k}$, but at this point the naive $O(n^2)$ algorithm is already preferrable.

Extension to mixed GIRGs Although algorithm 2 has a poor runtime on max GIRGs, it is easy to see that it fairs better for generating mixed GIRGs with a distance function that has an outer minimum - of the form

$$r(\boldsymbol{x}, \boldsymbol{y}) = \min\left(\|\boldsymbol{x}_{I_1} - \boldsymbol{y}_{I_1}\|, ..., \|\boldsymbol{x}_{I_k} - \boldsymbol{y}_{I_k}\|\right) \tag{2.26}$$

We just need a routine to sample the smaller-dim GIRGs in each dimension subset I_j (using the modified edge probabilities that conform to the resultant mixed GIRG), which also returns the distance in that subset for each edge. Then we can collectively consider each of these smaller-dim GIRGs' edges together to construct the final output GIRG. Therefore we can efficiently sample high dimensional mixed min/max GIRGs as long as each inner max dimension subset is small, such as $|I_j| \leq 3;\ j = 1, ..., k$.

2.5 Distorted GIRGs

A final GIRG variant that we only play with in chapter 4 is the *distorted GIRG*. This is a GIRG where the node locations' dimensions aren't all treated equally - rather they are ordered in importance $1 > 2 > ... > d$ via some weighting scheme $1 = \lambda_1 \geq \lambda_2 \geq ...\lambda_d > 0$. For instance a max norm distorted torus GIRG would have

$$r(\boldsymbol{x}, \boldsymbol{y}) = \max_{i \in [d]} \lambda_i |x_i - y_i|_{\mathbb{T}} \tag{2.27}$$

Oddly, to distort a MCD GIRG you instead have to use the inverse weights λ_i^{-1}, as now the smallest dimension distance has the greatest influence on the overall node distance.

The idea of distorted GIRGs is natural - that some dimension are more important than others in influencing edges. For social networks it could be e.g. that socioeconomic status of individuals has much higher impact on their friendships than personality type.

The weighted representation is easy to work with but an alternative is to literally distort the space $\chi = \mathbb{T}^d$ or $\chi = [0,1]^d$ to be cuboidal, i.e. no longer cube shaped. In order to remain consistent with our previous prior of node locations uniformly distributed in the cube, the node distribution in the cuboid would no longer be uniform. The simplest method would be to sample nodes first in the cube, and then map/rescale the whole space and locations together into a cuboid.

Chapter 3

Generative Graph Models (GGM) and Blasius Realism Framework

Contents

[Bläsius et al., 2018a] presents a general framework that can be used to assess the ability of a *generative graph model (GGM)* to simulate a set of real graphs of interest. They demonstrate their method on a few common GGMs: *Erdos-Renyi (ER)*, *Barabasi-Albert (BA)* preferential attachement graphs, *Chung-Lu* and *Hyperbolic* random graphs. The real graph dataset which they attempt to simulate is a set of 219 real-world networks taken from the *Network Repository* (`https://networkrepository.com/`, [Rossi and Ahmed, 2015]).

Their framework assesses realism of a GGM by using it to produce a dataset of fake graphs. A metric of distinguishability of fake and real datasets based on a select few graph statistics is produced: training a binary SVM classifier of graph

→fake/real using the combined fake and real graph datasets as training data, the accuracy of this classifier signifies real/fake distinguishability.

[Bläsius et al., 2018a] used their framework on a wider range of real-world graphs, but in particular found poor realism of the tested models on the ≈ 100 strong subset of Facebook social networks that we focus on. We will use the framework to specifically test different types of GIRGs' realism on this tough subset, with the hope that they will do better than the previous best models - Chung-Lu and Hyperbolic. We also think that restricting to this one subset of what should be similar types of real-world graphs will allow clearer differentiation between different GGMs.

3.1 GGM and Framework Mathematical Outline

A graph $G = (V, E)$ is said to be randomly generated from a GGM $\mathscr{G}$, written $G \sim \mathscr{G}$. Most of our GGMs are simple and can be described by a small number of parameters. For example for the Erdos-Renyi GGM, $G \sim \mathscr{G}_{ER}(n, p)$ means that there are n nodes: $V = [n]$, and each potential edge (u, v) exists iid with probability p, i.e. $p(u \sim v) = p$.

3.1.1 Fitting a GGM to a particular real graph instance

"Fitting" a GGM to a real graph G means finding the most plausible $\hat{\theta}$ in the hypothetical world where G was produced via $G \sim \mathscr{G}(\hat{\theta})$. Hence for our ER GGM example, choosing $\hat{n} = |V|$ is a no-brainer, and $\hat{p} = |E|/\binom{n}{2}$ follows by fitting the expected number of edges.

Fitting $\hat{\theta}$ can also be done by likelihood estimation. This is tractable for the Erdos-Renyi GGM, by solving $\arg\min_p \sum_{u \neq v} e_{uv} \log(p) + (1 - e_{uv}) \log(1 - p)$, where e_{uv} is the edge indicator function, which gives the same $\hat{p}$ as above. For GIRGs however, this would be intractable: for example we cannot even easily integrate over all possible point locations in the geometric space:

$$p(G|\alpha, \boldsymbol{w}, d) = \int_{\boldsymbol{x}_1 \in [0,1]^d} \cdots \int_{\boldsymbol{x}_n \in [0,1]^d} p(G|\alpha, (\boldsymbol{x}_i)_{i=1}^n) \prod_{i=1}^n p(\boldsymbol{x}_i) \tag{3.1}$$

Instead we can use a heuristic method based on *Approximate Bayesian Computation (ABC)*. We seek the parameter $\hat{\theta}$ that minimises the expected distance $\mathbb{E}[\delta(G, G' \sim \mathscr{G}(\theta))]$ for some distance metric δ. This basically means that the GGM parametrised by $\hat{\theta}$ successfully generates graphs G' that look similar to our given graph G. For δ to be effective, ideally it would be something like $(f(G) - f(G'))^2$, for f a sufficient statistic of θ. In practice this looks like an algorithm where we repeatedly sample θ from a prior; use it to generate G'; see if $f(G') \approxeq f(G)$, and if so keep θ as a candidate for $\hat{\theta}$.

3.1.2 Plausibility of Fitted GGMs

Similarly to the use of a statistic f of real and generated graphs as a proxy for fitting $\hat{\theta}$, we can also use other high level statistics f^{class} to evaluate the plausibility

of a fitted GGM as to whether it could have produced the real graph G. We will later use these as the input features to the real/fake binary classifier, hence the notation.

Having fit $\hat{\theta}$ of the GGM, we can then sample graphs $G' \sim \mathcal{G}(\hat{\theta})$ which are hypothetically similar to G. We can only hope for similarity on the global level. Even in the best case scenario where both $G, G' \sim \mathcal{G}(\hat{\theta})$, there is no local identification of nodes and edges, e.g. we cannot expect $e \in E \iff e \in E'$ - see appendix A.1 for more on graph similarity.

For example in the Erdos-Renyi exact fit case, in the best case scenario we have $G \sim \mathcal{G}_{ER}(n, p^*)$ and $G' \sim \mathcal{G}_{ER}(n, \hat{p} = p^* + \varepsilon)$. Although there will be some small estimation error ε in the fit parameter (which may be biased to be non-zero if e.g. G randomly has $|E|$ differing from $p^*\binom{n}{2}$), we've explicitly fit for the high level statistic of the number of edges, s.t. $|E| = \mathbb{E}[|E'|] = \hat{p}\binom{n}{2}$, and $|E'| \approxeq |E|$ with some small variance. For other (not fit) high level statistics f^{class} like the effective diameter of the graph, their expected value will be close to that of G, $E_{G' \sim \mathcal{G}(\hat{\theta})}[f^{\text{class}}(G')] \approxeq f^{\text{class}}(G)$, and likewise concentrated with small variance s.t. $f^{\text{class}}(G') \approxeq f^{\text{class}}(G)$.

Furthermore if we were to generate a whole list of graphs $G'_1, \ldots, G'_m$ from $\mathcal{G}(\hat{\theta})$, assuming the matching GGM hypothesis, we would not expect G to stand out from the crowd. Essentially the random variable $f^{\text{class}}(G'(G))$ should follow a distribution with $f^{\text{class}}(G)$ not too far from the mean/median, in an empirical p-value sense.

3.1.3 Fitted GGMs as candidates for real graphs - Blasius

[Bläsius et al., 2018a] takes this method one step further. They are essentially evaluating the hypothesis that the set of real graphs $G_1, ..., G_k$ are generated from a particular GGM $\mathcal{G}$, but with differing parameters: $G_1 \sim \mathcal{G}(\theta_1^*), ..., G_k \sim \mathcal{G}(\theta_k^*)$. Perhaps θ_i^* come from some prior distribution $p(\theta)$.

They again fit individual $\hat{\theta}_i$ with which to generate one $G'_i \sim \mathcal{G}(\hat{\theta}_i)$ per real graph G_i. This way instead of, for each real graph, comparing multiple G'_{ij} to G_i and averaging, they can compare the set of graphs $\{G_i\}_{i=1}^k$ to $\{G'_i\}_{i=1}^k$. Graph comparison is done by comparing a wide number of statistics/metrics computed for each graph. These are input as a whole feature vector $\boldsymbol{f}^{\text{class}}(G)$ to an SVM classifier, which is trained to classify membership of the real or generated dataset. These features are statistics such as the number of nodes, mean node degree, mean node closness centrality, graph diameter and so on.

Performing this classification task over the whole dataset of graphs allows for a GGM realism evaluation without needing to have the additional computation of generating multiple G'_{ij} for each real graph G_i, which would have given a more detailed graphwise GGM realism metric. There is an added benefit of helping to cover for GGM parameter fitting inaccuracy. If $\hat{\theta}_i = \theta_i^* + \varepsilon_i$, it may still be possible to distinguish G_i from $\{G'_{ij}\}_{i=1}^m$ due to the ε_i fitting error. If instead we compare the set $\{\theta_i^*\}_{i=1}^k$ to $\{\hat{\theta}_i\}_{i=1}^k$, Now each $\hat{\theta}_i \neq \theta_i^*$, but still well fits into the distribution of $\{\theta_i^*\}_{i=1}^k \sim p(\theta)$. Finally having one G'_i per real graph G_i also simplifies the binary classification task by having a balanced dataset.

The output of the Blasius framework, for a given GGM and classification feature vector f^{class}, is a real/fake accuracy percentage. If it were really the case that the input real-graphs were generated from $G_i \sim \mathcal{G}(\theta_i^*)$ of the same GGM $\mathcal{G}$ as the fitted fake graphs, the SVM should be able to acheive only close to 50% accuracy, as real and fake f^{class} should be hard to tell apart. Higher accuracies denote distinguishability of real from fake - occuring if instead $G_i \sim \widetilde{\mathcal{G}}(\phi_i^*)$ for some alternative GGM $\widetilde{\mathcal{G}}$.

3.2 Fitting GIRG GGM to a real graph

The GGM parameters of the GIRG model $\mathcal{G}_{\text{GIRG}}(\theta)$ are $\theta = (n, d, c, \alpha, \tau)$. For the Blasius framework we need to fit $\hat{\theta}$ to a given real graph G, as the best estimate for the parameters that would have generated $G \sim \mathcal{G}_{\text{GIRG}}(\theta)$. The problem of fitting dimension d is a tricky one that we touch on briefly in chapter 4. We also attempted to use *graph fractal dimension* to infer d without success. We instead avoid the problem by treating 1d, 2d, 3d etc. GIRGs as separate GGMs, to be individually fitted and inter-compared.

3.2.1 Fitting number of nodes n

We follow [Bläsius et al., 2018a] which first preprocesses the graph G by shrinking it to its largest connected component, $G \leftarrow \text{shrinkToGCC}(G)$ before fitting $\mathcal{G}$ to it. Then n is just the number of remaining nodes.

Blasius' rational is that where some real graphs in our dataset have disconnected subgraphs, this may be due to them being a concatenation of a few distinctly generated subraphs, hence a GGM should be fit just to one subgraph (the largest - the rest are discarded for potentially being too small). All of our GGMs are capable of producing a bunch of disconnected subgraphs, however this is most likely to occur in our geometric GGMs which exhibit clustering; GIRGs do at least whp produce a unique giant component - one with a linear number of nodes. Hence for a fair comparison between G and G', we also need to post-process the GGM generated $G' \leftarrow \text{shrinkToGCC}(G')$.

Blasius's only geometric model is Hyperbolic Random Graphs, for which they use a fitting algorithm that actually estimates a higher number of nodes $n > |V|$ in order to approximately have the largest connected component of $G' \sim \mathcal{G}$ be of size $|V|$. We don't do this for our GIRGs however, as we find this algorithm prone to error, and unnecessary, at least for the socfb Facebook graphs - see table 3.1. We accept instead that our GGMs may end up with slightly fewer nodes than the real graphs, and hope that this doesn't affect the SVM classification too much.

Restricting graphs to a connected component also has the added benefit of making some graph statistics more meaningful/sensical - for instance diameter and path lengths.

Note that GIRGs might still do the best job, compared to other non geometric GGMs, of fitting multiple large components, as they have the property of containing sublinear separators. Essentially if you divide the torus with a hyperplane (or divide out a sub-cube), $\chi = \chi_A \sqcup \chi_B$, to produce disjoint node subsets $V_A \sqcup V_B = V$ and their induced subgraphs G_A, G_B, then whp G_A has sublinear of $|V_A|$ edges to

graph name	GGM	nodes
socfb-American75	real-world	6370
socfb-American75	1d-girg	6370
socfb-American75	2d-girg	6370
socfb-American75	3d-girg	6370
socfb-American75	ER	6370
socfb-American75	chung-lu	6279
socfb-American75	hyperbolic	6583
socfb-Amherst41	real-world	2235
socfb-Amherst41	1d-girg	2235
socfb-Amherst41	2d-girg	2235
socfb-Amherst41	3d-girg	2235
socfb-Amherst41	ER	2235
socfb-Amherst41	chung-lu	2221
socfb-Amherst41	hyperbolic	2282
...	...	...
bio-diseasome	real-world	516
bio-diseasome	1d-girg	258
bio-diseasome	2d-girg	491
bio-diseasome	3d-girg	496
bio-diseasome	ER	512
bio-diseasome	chung-lu	459
bio-diseasome	hyperbolic	125

Table 3.1: For the socfb Facebook graphs, the hyperbolic model consistently has a few more nodes than the input real graph due to its fitting algorithm not quite working perfectly (and stochasticity). On other real graphs there can be larger discrepancies, especially for smaller extra sparse graphs. Numbers of nodes in the output (shrunk to GCC) graph are colored cyan if less than the real-world graph, and orange if more (only possible in hyperbolic case).

G_B (despite having linear number of edges within itself). Also interestingly G_A is stochastically still a (smaller) GIRG, just one with a non-toroidal geometry.

3.2.2 Fitting power law distribution exponent τ for node weight sampling

We fit τ to the tail of the *degree distribution* of G, using the python package `powerlaw`. Having this fit parameter enables subsequent generation of new weight sequences $w_u \sim \text{powerlaw}(\tau)$ for sampling new GIRGs $G' \sim \mathcal{G}_{\text{GIRG}}(\tau, ...)$.

The degree distribution of the graph G is defined as

$$dd(x) := \frac{|\{v \in V : d(v) = x\}|}{|V|} \tag{3.2}$$

i.e. the fraction of nodes with degree x. For graphs generated by the GIRG model, we noted that the geometry marginalisation property means that $E[d_u] \propto w_u$. Hence if node weights are distributed following $\text{powerlaw}(\tau)$, we expect to see the tail of the degree distribution, $dd(x)$ as $x \to n$ to look like a discrete power law distribution $dd(x) \propto x^{-\tau}$.

Informal sketch of why GIRG power law weights → power law degree distribution By treating node degrees as roughly iid, $dd(x) \overset{n\to\infty}{\to} P(d_u = x)$. For a fixed weight w_u, node degree d_u closely follows a binomial distribution $\text{binomial}(n-1, \Theta(w_u/n))$, so we'd like to calculate

$$P(d_u = x) = \int P(d_u = x | w_u = w)\Theta(w^{-\tau})\, dw \tag{3.3}$$

For sufficiently large x (in the tail of the decaying distribution of $dd(x)$), the binomial distribution of mean and variance w is sharply peaked around its mean: if $(x - w) > w^{1/2+\varepsilon}$ then $P(d_u = x | w_u = w) \approx 0$. So essentially all the probability mass is on the region $[x - x^{1/2+\varepsilon}, x + x^{1/2+\varepsilon}]$, on which $\Theta(w^{-\tau}) \approx \Theta(x^{-\tau})$ (doesn't change much at all) as e.g. $(x - x^{1/2+\varepsilon})^{-\tau} = (x[1 - x^{-1/2+\varepsilon}])^{-\tau} \approxeq x^{-\tau}$ for large x. Hence $dd(x) \overset{n\to\infty}{\to} \Theta(x^{-\tau})$ for large x.

Python powerlaw package Therefore if G is a τ exponent power law weighted GIRG , we expect its degree distribution $dd(x)$ for large degrees x to look like $dd(x) \propto x^{-\tau}$. Hence we can fit τ on this tail. `powerlaw` does this by simultaneously finding a lower bound $x_{\min}$ for where in the tail the power law behaviour starts, and fitting τ to the region $x > x_{\min}$ using maximum likelihood estimation[1]. The optimal $x_{\min}$ is chosen to minimise the Kolmogorov-Smirnov distance between the power law fit and the region's emprirical degree distribution.

Having fit τ, we can then sample weights $w_u \sim \text{powerlaw}(\tau)$.

3.2.3 Power Law weight alternative: weight copying

Generating weights $w_u \overset{iid}{\sim} \text{powerlaw}(\tau)$ is fine as a model prior, however it's not a perfect fit to real-world data. This is particularly true as real graph degrees generally only follow a power law for large degrees; they might be better modelled by a GIRG with node weights with a power law tail (e.g. upper quartile of weights) and a different distribution for the rest of the weights. Most obviously, instead of having peak number of nodes with weight $w_u = x_{\min}$, the weight distribution might follow a more natural curve like in fig. 3.1.

An easy improvement to better fit a specific real graph is to take the sequence of node degrees as weights [2]. Copying degrees as weights makes sense due to the Chung-Lu like property of having $E[d_u] \propto w_u$, whose correct scaling can be made correct by fitting c as detailed in the next section. For the classification comparison framework, Blasius actually uses weight copying for fitting the Chung-Lu GGM, but not for the hyperbolic GGM (which is odd). This doesn't make a fair / like for like comparison, so we cover all bases by having both weight copied and power law fit GIRGs, as well as adding in a power law fit Chung-Lu. ER and BA GGMs are still just fit on the courser metric of overall graph average degree, as they have little ability to replicate a power law degree distribution.

[1] A visually much more compelling method of fitting the power law degree distribution tail is to plot the empirical degree distribution on a log-log scale, and literally draw a line of best fit on the heavy tail - $w^{-\tau}$ will look like a straight line with gradient $-\tau$. In fact one minus the empirical cdf is even better as it looks like $w^{1-\tau}$, so a similar log-log linear fit works to find τ, and the plotted points are more robust to noise

[2] these are now all positive integers ≥ 1 as opposed to real numbers $\geq x_{\min}$, since in the largest connected componenet, minimum degree is 1

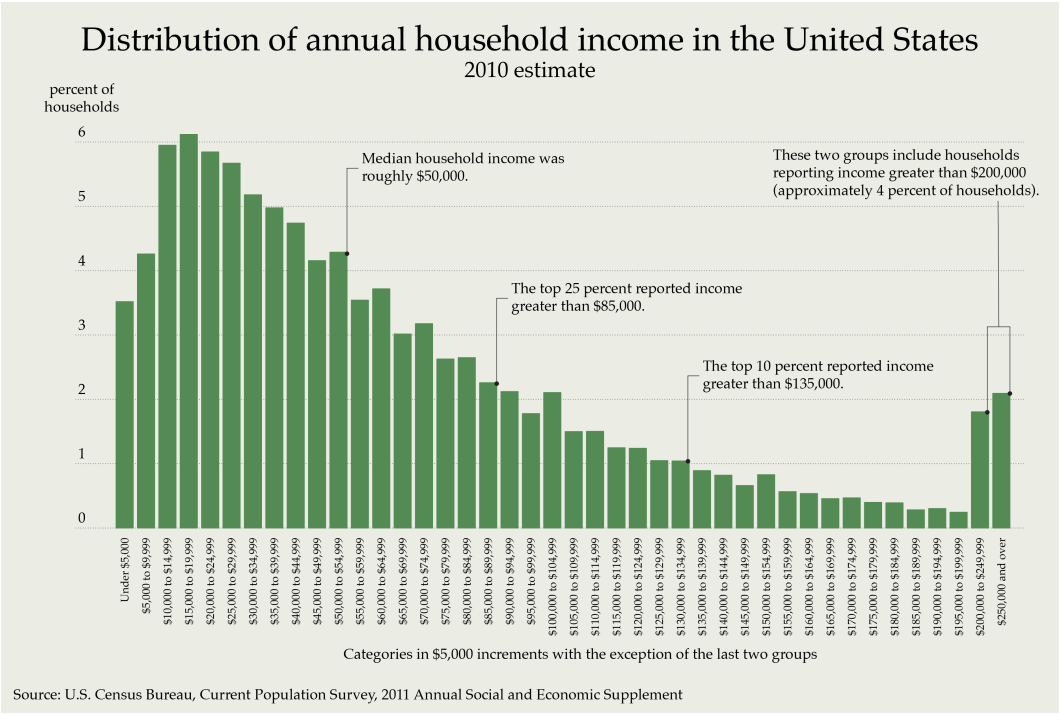


Figure 3.1: Example of a more natural weight distribution with a power law tail but not for smaller weights.

Comparing weight copied Chung-Lu with weight copied GIRGs was additionally interesting for harder feature sets where other GGMs struggled to get below 98% classification accuracy - having such high constrained numbers makes it harder to compare meaningfully between GGMs (drawing conclusions from 98% vs 99% is tough)

3.2.4 Fitting c and α

These are our last two GIRG parameters to fit. The parameter $c \in \mathbb{R}^+$ is most directly linked to the generated graph's average degree, whereas the parameter $\alpha \in (1, \infty]$ is more linked to the power of the geometry - larger α decreases the probability of longer distance edges which have $\rho_{uv} < 1$, leaving the edge set more dominated by shorter (and hence more geometric and clustered) edges. We therefore follow [Bläsius et al., 2018a] by fitting α with the clustering coefficient. Regrettably we used in our experiments the mean *local clustering coefficient (LCC)* to fit our GIRGs and Hyperbolic Random Graphs,

$$\mathrm{LCC}(G) := \frac{1}{|V|} \sum_{u \in V} \frac{|\{v, v' \in \Gamma(u) : v \sim v'\}|}{\binom{|\Gamma(u)|}{2}} \in [0,1] \tag{3.4}$$

instead of the global clustering coefficient as in [Bläsius et al., 2018a]. $\Gamma(u)$ is the set of neighbours of u, so LCC counts the fraction of pairs of neighbours of u that possess the triangle completing third edge. The global clustering coefficient is the fraction of total "V" shapes that indeed are completed as full triangles and is likely a better metric than the LCC. Luckily in practice they generally differ by less than 1%.

Unfortunately c and α are not wholly independent, so we must fit them in tandem, fitting c for a given α so as to match the average degree of G, and then α for that

given c to match the LCC of G. This then looks like coordinate ascent in 2D: we alternatingly set $c \leftarrow \hat{c}_1$; $\alpha \leftarrow \hat{\alpha}_1$; $c \leftarrow \hat{c}_2$; $\alpha \leftarrow \hat{\alpha}_2$; ....

Fitting $\alpha \leftarrow \hat{\alpha}_i$ based on LCC is precisely ABC like - we propose potential α_i values, for which we generate a graph $G' \sim \mathcal{G}_{\text{GIRG}}(n, d, \hat{c}_i, \alpha_i, \tau)$, and use the distance metric $\delta(G, G') = |\text{LCC}(G) - \text{LCC}(G')|$ to both propose a next candidate α_i and eventually accept the final best $\hat{\alpha}_i$. Expected mean LCC is monotone in α so we know which direction to make a new proposal, and we follow [Bläsius et al., 2018a] in doing binary search in α^{-1} search space with starting bounds $[0.1)$.

[Bläsius et al., 2022] luckily gives a more efficient method to fit c given α, d, and a pre-sampled set of weights $(w_u)_{u \in V}$, that doesn't involve fully sampling a new $G' \sim \mathcal{G}_{\text{GIRG}}$. They derive a formula for the expected average degree $\mathbb{E}[\overline{deg}(G')]$ of the GIRG which looks, emphasizing the dependence on c, like:

$$\mathbb{E}[\overline{deg}(G')|c] = cA + c^{1/\alpha}B \tag{3.5}$$

Hence we just have to numerically solve eq. (3.5) for

$$\hat{c}: \ \mathbb{E}[\overline{deg}(G')|\hat{c}] = \overline{deg}(G) = 2|E|/|V| \tag{3.6}$$

in order to obtain our desired average degree. Bläsius provides C++ code for this, which finds $\hat{c}$ with a binary search.

The expected average degree formula of eq. (3.5) miraculously holds for all volume based toroidal GIRGs (regardless of exact disance function $r(x_u, x_v) = r_{uv}$), and we can even adapt the formula to be independent of dimension d.

3.2.5 Volume formulation torus GIRG expected average degree formula

In this subsection we calculate $E_r[p_{uv}(r)|c]$ for fixed weights w_u, w_v in a volume formulated GIRG, whose volume function is denoted $Vol(r)$ (could correspond to any of max norm, euclidean norm, MCD etc.), and arbitrary dimension d torus geometry $\chi = \mathbb{T}^d$.

Given a fixed weight sequence $(w_u)_{u \in V}$ we will recover eq. (3.5) via

$$E[\overline{deg}(G')|c] = \frac{1}{n} \sum_{u \in V} \sum_{v \in V} E_r[p_{uv}(r)] \tag{3.7}$$

Now $E_r[p_{uv}(r)] = \int_r p_{uv}(r)p(r)dr$. We'll break this down into $\int_{r:\rho_{uv} \geq 1} + \int_{r:\rho_{uv}<1}$ (short edges and long edges, corresponding to edge probabilities being capped at 1, or strictly less than one for larger inter node distances) and write $\hat{r} : Vol(\hat{r}) = c^{1/\alpha}\left(\frac{w_u w_v}{W}\right)$ as the boundary at which $\rho_{uv} = 1$.

Hence we get

$$E_r[p_{uv}(r)] = \int_0^{\hat{r}} p(r)dr + \int_{\hat{r}}^{r_{\max}} p_{uv}(r)p(r)dr \tag{3.8}$$

Substituting $Vol = Vol(r)$; $dr = dVol \frac{dr}{dVol} = \frac{dVol}{p(r)}$, we get:

$$\begin{aligned} E_r[p_{uv}(r)] &= \textstyle\int_0^{Vol(\hat{r})} dVol + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} p_{uv}(Vol)dVol && (3.9)\\ &= Vol(\hat{r}) + \textstyle\int_{Vol(\hat{r})}^{Vol(\mathbb{T})} c\left(\frac{w_u w_v}{W}\right)^{\alpha} Vol^{-\alpha} dVol && (3.10)\\ &= Vol(\hat{r}) + c\left(\frac{w_u w_v}{W}\right)^{\alpha} \frac{1}{\alpha - 1}\left[\left(\frac{w_u w_v}{W}\right)^{1-\alpha} c^{\frac{1-\alpha}{\alpha}} - 1\right] && (3.11)\\ &= c^{1/\alpha}\left(\frac{w_u w_v}{W}\right)\left[1 + \frac{1}{\alpha - 1}\right] - \frac{c}{\alpha - 1}\left(\frac{w_u w_v}{W}\right)^{\alpha} && (3.12) \end{aligned}$$

where in the last two lines we sub in $Vol(\hat{r}) = c^{1/\alpha}\left(\frac{w_u w_v}{n}\right)$, and $Vol(\mathbb{T}) = 1$. I.e. across all GIRGs of different distance functions and dimensions d, we get the same resultant edge probabilities when marginalising out node locations. Note this is also $\Theta(\frac{w_u w_v}{W})$ since $\alpha > 1$, which shows the similarity between GIRGs and Chung-Lu!

Following [Bläsius et al., 2022] Appendix, the formula just needs a correction for pairs u, v such that $Vol(\hat{r}) = c^{1/\alpha}\left(\frac{w_u w_v}{n}\right) > 1$, for which the second integral is unnecessary, and the first's upper bound is capped lower at 1, the volume of the torus. These are (rare) pairs of nodes u, v which have such giant weights w_u, w_v that $p_{uv}(r) = 1$ identically, no matter how far apart x_u, x_v are placed in the torus.

3.2.6 Estimating const c in Cube GIRGs

To fit c for a specific real graph G, for torus GIRGs we solved for the equation $f(c) = \overline{deg}$, given fixed α. Our distance based Cube GIRGs have fewer edges in expectation than their toroidal counterparts, and no nice formula for the expected average degree. Instead we estimate $\hat{c}$ by starting with an initial guess $c_0 = 1$, and iteratively updating by each time generating a graph $G_i \sim \text{GIRG}(c_i)$ and setting $c_{i+1} \leftarrow c_i \frac{n\,\overline{deg}}{2|E(G_i)|}$ until convergence.

3.3 Realism Framework Results

As introduced in section 3.1.3, we fit a selection of GGMs, including various different kinds of GIRG on 104 socfb Facebook graphs, whose sizes range from $n = 762$ to $n = 35,111$ nodes. With each GGM we generated a mirrored dataset of fake graphs, with which we train a sequence of SVM classifiers to distinguish between the real and fake datasets, differing on which high level graph features they use as input. Classification accuracy using a particular GGM and feature set could be e.g. 100% if the fake graphs are highly unrealistic in these features, or as good as 50% if they're indistinguishable from the real graphs. Our results are shown in fig. 3.5 and fig. 3.7, along with a "control" in fig. 3.8 which classifies GGM fake datasets against the 1d max torus generated graphs instead of the real graphs. The numbers displayed in all three are actually the *misclassifcation rate*, i.e. $1 - \text{accuracy}$, so that higher numbers mean more realistic GGM fit.

In fig. 3.5 we present results of the harder classification test of using a whole distribution of node-level features. E.g. one node level property "deg" actually means the inclusion of 5 features describing the distribution of node degrees: mean, median, lower quartile, upper quartile and standard deviation. In contrast the classification results in fig. 3.7 and fig. 3.8 just use the mean node level features, which makes it easier to fool the classifier. Having more significantly higher than zero misclassification rate numbers in fig. 3.7 allows for more meaningful comparisons between different GGMs and across a wider range of features.

Types of GIRG tested in classification framework

- The "base" GIRG type is max norm torus GIRGs of dimension $d = 1, 2, ..., 7$
- MCD torus GIRGs for $d = 2, 3, 4, 5$ ($d = 1$ is equivalent to max norm)

- Min/max mixed torus GIRGs with groupings 1-23; 1-234; 12-34; 1-2-34. E.g. 1-23 denotes $\|\boldsymbol{x}-\boldsymbol{y}\| = \min\{|x_1-y_1|_{\mathbb{T}}, \max(|x_2-y_2|_{\mathbb{T}}, |x_3-y_3|_{\mathbb{T}})\}$
- Max norm cube GIRGs. We were held back due to the increased computation when fitting the average degree with constant c, so only include $d = 1,2,3$ cube GIRGs, and $d = 1,2,3,4,5$ copy-weight cube GIRGs (an attempt for maximal realism)
- The host of simpler models CL (with power law fit weights, and with degree copied weights), BA, ER as in [Bläsius et al., 2018a], as well as HRGs which are very similar to 1d torus GIRGs

New classification features We additionally added features that were not assessed in [Bläsius et al., 2018a]. Shown in fig. 3.5 are the non-node-level features "k-cores, comms". These denote k-core sizes and community sizes (communities found using the Louvain method). However these can be rather high variance numbers. For example a 3000 node 2d max torus GIRG with average degree of 60 can end up with 40 k-cores and just 8 communities. This means that the community size distribution features would be the average size of these 8 communities, as well as lower, middle and upper quartile size and standard deviation of these sizes.

Unfortunately, for community sizes, even Erdos-Renyi graphs have a misclassification rate comparable to the other GGMs, so this feature seems useless.

3.3.1 Degree distribution features

As noted in [Bläsius et al., 2018a] features like Katz centrality, PageRank, k-core number / k-core sizes, node degrees and node degree power law are more closely related to the degree distribution. Consequently they are less useful for distinguishing between different types of GIRG geometries.

For this group of features we naturally observe much better performance for copy-weight GIRGs / copy-weight Chung-Lu which have the most similar degree distributions to the real graphs - Chung-Lu doing the best as it has the purest degree alignment. For all these features except "n, m, Katz" in fig. 3.5 (distribution features), non copy-weight GGMs have all zero misclassification rates. For k-core, deg and Katz, the copy-weight GIRGs also feature monotonically increasing misclassification rates with increasing dimension, presumably as higher dimensions make the GIRG more similar to the Chung-Lu - see [Friedrich et al., 2023].

For just mean features in fig. 3.7, there are many more non-zero rates for non-copy-weight GGMs, but generally worse numbers than the copy-weight GGMs. For whatever reason, non-copy-weight 3d cube GIRGs are much better on k-core (16%) than all other GGMs, even copy-weight 3d cube GIRGs which get just 7%, and decently better on Katz centrality (17%) than the other non-copy-weight GGMs. This could be due to a more realism of extreme nodes near the edge of the cube that both have a lowered degree, and neighbours which are also similarly low degree due to being extremely located.

3.3.2 Geometric Features

The features betweenness centrality, closeness centrality, local clustering coefficient, diameter and effective diameter are more related to the geometry. GIRGs show on

these features increased realism in comparison to the non geometric GGMs, and we can also compare between the different GIRG types: max / MCD / mixed distance functions; torus / cube geometries. We focus exclusively on the non-copy-weight GIRGs, with numbers from the mean based classifiers in fig. 3.7 rather than the distribution based classifiers in fig. 3.5 which have too low numbers for a good comparison. In the former, the copy-weight cube GIRGs have similar performance on these features - their more realistic degree distribution only help in the latter, more challenging simulation task.

Chung-Lu graphs are a good comparison point against GIRGs by being a similarly degree inhomogenous graph, just without any geometric influence. [3].

Among non-copy-weight GIRGs, 1d max torus GIRGs are surprisingly best or close to best on many feature sets. This is evidence that the geometric component of real graphs is strongest in just one dimension, which we also see in chapter 5. It would be interesting to see if 2d or 3d distorted GIRGs could do better on these features, as they would precisely exhibit geometry with one dominating dimension, and a few auxiliary ones. On betweenness, 1d max torus GIRGs perform the best, with a slight decay for 2d-4d. Mixed GIRGs hold up well, but MCD GIRGs do much worse; cube GIRGs are only a little worse. However MCD and mixed GIRGs do better on closeness than max GIRGs, especially 1-23 mixed GIRGs. This is actually evidence for the "one strong dimension" hypothesis, as 1-23 mixed GIRGs give more significance to their dim 1 over dims 2,3. We can even observe in fig. 3.8 that 1-234 mixed GIRGs have such a strong force from their first dimension that they are hard to classify apart from 1d max torus GIRGs!

Effective diameter and cube GIRGs Cube GIRGs generally underperform, except on effective diameter, where they shine through as the best, with ranking 1cu<2cu<3cu. This makes sense as cube geometry doesn't impact many shortest paths, but rather does so on the longest shortest paths, between nodes at opposite extremes of the cube which would otherwise be shortcutted by the torus. We can dive into the data to see that our intuition seems correct: real effective diameters of FB graphs range from 3.0 to 4.4; 2d torus GIRGs' are too small, on average 0.17 less than their real counterparts, while 2d cube GIRGs' are larger and on average just 0.04 less.

Local Clustering Coefficient (LCC) Due to their geometry, GIRGs exhibit significant clustering just like real graphs. We explicitly match for the mean LCC in our GIRG fitting procedure, so it's not surprising that the "n, m, LCC" row in fig. 3.7 has near-perfect performance for most GIRGs, whereas all other GGMs have a 0% misclassification rate as they have tiny mean LCCs. This drops to 0% for all GIRGs except copy-weight cube GIRGs when classifying based on LCC distribution. [Bläsius et al., 2018a] explained this by showing their fit hyperbolic graphs to have too low variance, which is likely the case for GIRGs here too.

[3] Another potential GGM geometry null hypothesis would have been the configuration model which tries to mimick a degree sequence, like Chung-Lu, but more strictly. Each node is given a set number of "half edges" which gives it a set degree (the degree sequence can be random e.g. power law generated, or given e.g. copied directly from a real graph). Half edges are then joined together at random.

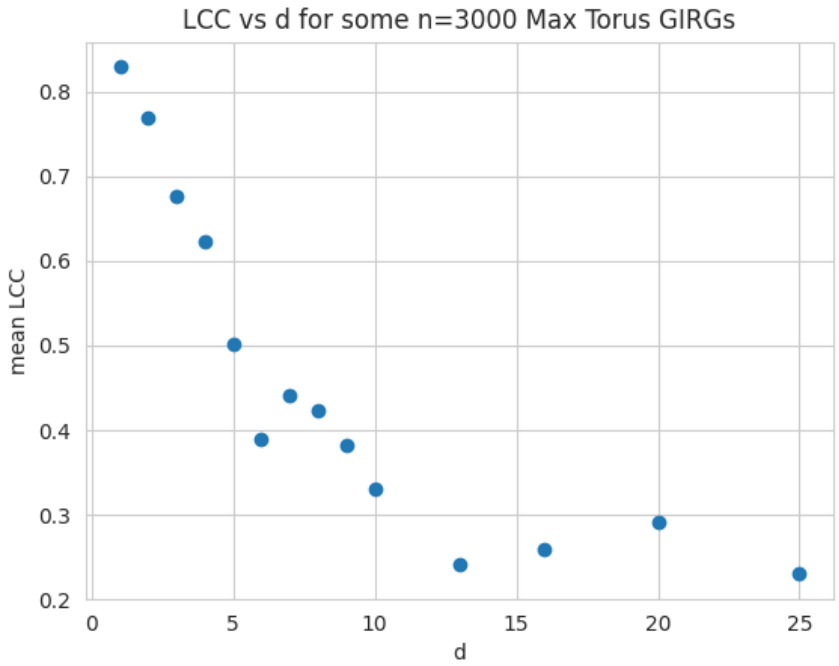


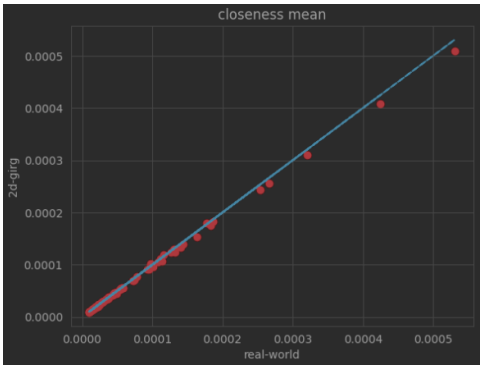
Figure 3.2: Increasing the dimension d of a GIRG will decrease the mean LCC, but only up to a $\Theta(1)$ threshold due to the triangle inequality. Each point is one generated GIRG, all other parameters remaining fixed.

For mean LCC only 4d, 5d MCD GIRGs and 6d, 7d max GIRGs exhibit low misclassification rates, due to their inability to match the real graphs' high mean LCC. E.g. on socfb-Caltech36, the real graph has a mean LCC of 0.41, however capping at $\alpha = 100.0$ ($\alpha = \infty$ wouldn't improve the situation much either), the max norm 4d torus GIRG can only reach 0.33, and the 5d GIRG just 0.24. LCC is guaranteed to be non-tiny in GIRGs due to the triangle inequality: if $u \sim v_1$ and $u \sim v_2$ then likely $r(u,v_1), r(u,v_2)$ are both small, which makes $r(v_1,v_2) \leq r(u,v_1) + r(u,v_2)$ also small. However for higher dimension GIRGs, $r(v_1,v_2)$ is more likely to be closer to this upper bound. LCC decays already for 4d MCD GIRGs as the MCD distance function isn't actually a metric so only obeys the triangle inequality stochastically (some percentage of the time. E.g. if u is close to v_1 in dimension 1, and close to v_2 in dimension 2 then v_1, v_2 are generally not close to each other).

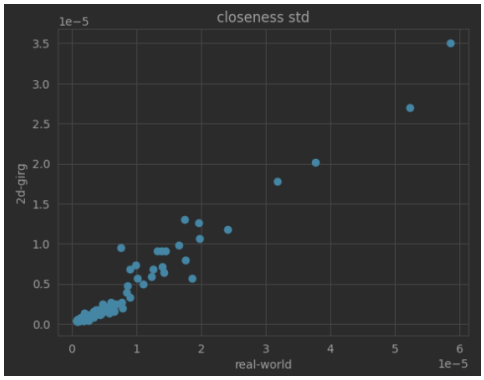
3.3.3 Case study of distribution vs mean based classification: closeness centrality

In fig. 3.5 which provides feature distributions as input to the SVM classifier, it is disappointing that non copy-weight GIRGs show such poor performance, even on "geometric" features like closeness, betweenness and LCC (at least on closeness and betweenness they still recognizably outperform CL, BA and ER). E.g. 2d max torus GIRGs have a misclassification rate of just 1% on "n, m, betw" and 2% on "n, m, close", compared to much better 38% and 28% on the mean based classifier equivalents. To explain this large disparity we do a case study on closeness, which happily also assuages some of our concerns.

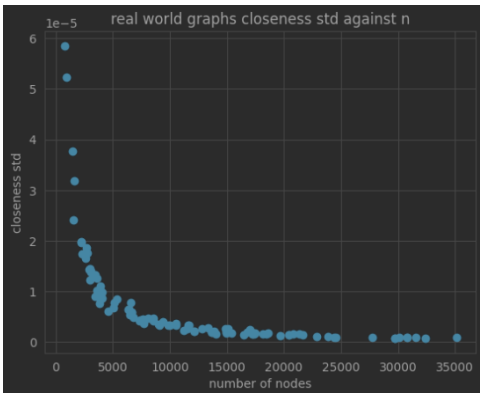
fig. 3.3a shows that mean closeness aligns very accurately to the real graphs. Nonetheless, due to a consistently lower standard deviation of closeness in the GIRGs and the fact that the standard deviation of closeness in real graphs fits



(a) 2d GIRGs have almost identical mean node closeness to real-world FB graphs; $y = x$ line shown in blue.



(b) 2d GIRGs have consistently lower standard deviation of node closeness than real-world FB graphs



(c) Standard deviation of node closeness in real-world FB graphs fits well as a function of number of nodes

Figure 3.3: Closeness centrality case study on fit 2d max torus GIRGs against their real counterparts. In plots (a) and (b), each dot is a real / fake graph pair, with the x/y value being the specifid feature's vale for the real/fake graph respectively. In plot (c), each dot is a real graph.

well as a function of number of nodes, the 2d GIRGs are distinguishable from real graphs on this feature set. We hence find the 2% misclassification rate misleadingly low, and the GIRG's realism in closeness somewhat vindicated.

3.3.4 One drawback to the classification realism framework

Classification accuracy is regrettably a crude metric. One limitation of this framework is that the attainable accuracy is also affected by the level of variance of a feature set across the real graph dataset relative to the fake dataset. GGMs generally end up with their features concentrated in a small subsection of the feature space, so if the real graphs are more spread out, it may be easier to overlap with a portion of their feature data points and attain a low but appreciable misclassification rate. However if the real graphs are less spread out they can miss altogether and get a minuscule misclassifcation rate. See fig. 3.4 for an illustration of different scenarios. In this case of closeness, the real graph feature vectors fit quite a precise pattern and hence the GIRG feature vector clump, though just slightly off, has little overlap.

3.3.5 SVM Classifier Misclassification Rates Tables

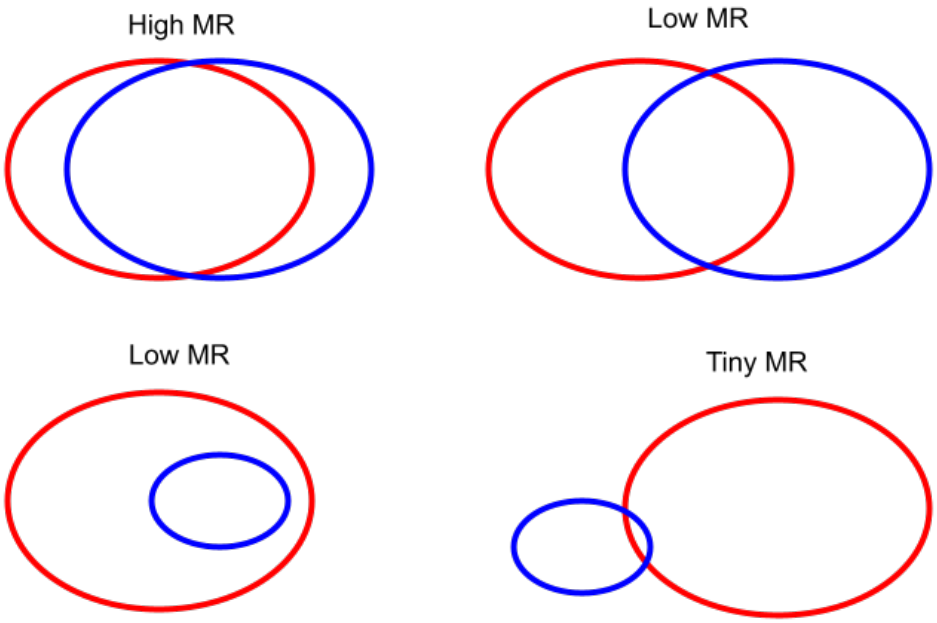


Figure 3.4: Illustration of SVM misclassification rates (MR) given different positions and variance of binary clusters. Red and blue ovals represent roughly gaussian distributed clusters of real and fake data points in a 2 dimensional feature space. An SVM would divide the space so as to maximise classification accuracy. How high accuracy it achieves is directly related to the area/density of overlap between the two clusters.

Feature Set	1ccu	2ccu	5ccu	CL-c	1-23	1-234	2m	5m	1cu	2cu	3cu	1d	2d	7d	CL	BA	ER	hyper
n, m, betw	18	16	5	5	1	1	2	0	1	0	0	1	1	1	1	0	0	1
n, m, close	8	15	16	10	1	0	0	3	2	3	5	2	2	2	1	1	0	3
n, m, LCC	4	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, diam	4	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, eff-diam	22	25	22	9	8	17	2	0	17	23	24	13	16	13	0	0	0	17
n, m, k-core	6	9	10	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, deg	14	23	34	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, Katz	25	25	36	45	6	6	6	9	7	3	1	6	7	5	3	0	1	6
n, m, PR	20	25	23	23	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, comms	3	4	4	9	2	2	1	1	6	6	6	5	3	11	10	5	8	1
n, m, k-cores	6	8	14	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.5: [Bläsius et al., 2018a] GGM realism framework results on Facebook graphs, extended to different GIRG models. Shown here are distribution based features, i.e. including mean, median, lower quartile, upper quartile and standard deviation of node level features. Some models are missing from the table e.g. 3d-6d GIRGs as their results follow a trend from low to high dimension.

1ccu	1d copy-weight cube GIRG	betw	node betweenness centrality
1-23	$1 \vee (2 \wedge 3)$ mixed min/max GIRG	k-core	node k-core number
2m	$1 \vee 2$ 2d min GIRG	close	node closeness centrality
3cu	3d cube GIRG	LCC	node local clustering coefficient
7d	7d GIRG	deg	node degree
CL	Chung-Lu	Katz	node Katz centrality
CL-c	copy-weight Chung-Lu	PR	node PageRank
BA	Barabasi-Albert	comms	community sizes
ER	Erdos-Renyi	k-cores	k-core sizes
hyper	Hyperbolic Random Graph	diam	graph diameter
		eff-diam	effective graph diameter

Figure 3.6: Graph Generative Model abbreviations; feature name abbreviations. Used in our results tables

Features	1ccu	2ccu	3ccu	4ccu	5ccu	CL-c	1-2-34	1-23	1-234	12-34	2m	3m	4m	5m	1cu	2cu	3cu	1d	2d	3d	4d	5d	6d	7d	hyper	CL	BA	ER
betw, close, diam	7	5	3	5	6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
betw, close, eff-diam	28	29	30	23	28	15	3	11	17	3	3	3	1	0	16	19	23	17	15	14	13	12	10	13	14	1	1	1
LCC, tau, diam	6	8	5	4	6	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
LCC, tau, eff-diam	27	26	31	31	31	0	6	17	19	5	5	3	1	0	22	27	26	19	20	20	18	19	1	0	19	0	0	0
n, m	49	48	48	48	48	48	49	50	49	50	49	48	47	47	48	50	49	50	50	50	50	49	49	49	46	50	50	50
n, m, betw	39	38	38	36	32	9	16	34	37	15	14	11	9	7	37	32	23	39	38	38	35	28	27	26	34	2	2	3
n, m, betw, close, diam	5	5	3	3	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, betw, close, eff-diam	21	25	22	21	21	9	2	8	16	2	2	1	1	1	14	18	18	16	16	14	13	12	9	14	16	1	0	2
n, m, close	33	33	32	34	27	10	29	42	35	35	35	20	16	11	24	20	17	31	28	29	21	16	24	42	17	8	2	12
n, m, close, LCC	38	39	37	36	29	0	39	47	36	39	40	27	5	0	26	21	22	36	29	26	22	16	1	0	24	0	0	0
n, m, close, deg	40	40	36	35	28	14	37	47	37	37	38	28	20	17	30	21	21	36	32	32	24	18	26	47	23	14	7	16
n, m, LCC	48	47	46	47	47	0	47	50	49	49	49	37	6	0	49	50	48	49	50	49	50	34	1	0	45	0	0	0
n, m, LCC, betw	35	36	36	36	35	0	16	38	39	18	16	11	3	0	41	32	24	43	39	38	35	17	1	0	37	0	0	0
n, m, LCC, tau	27	28	34	36	39	0	44	37	36	44	43	39	7	0	35	31	29	35	38	41	40	33	1	0	35	0	0	0
n, m, LCC, tau, diam	4	2	5	2	5	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
n, m, LCC, tau, eff-diam	21	18	22	19	19	0	2	11	17	2	2	1	0	0	22	26	25	15	14	17	19	18	1	0	18	0	0	0
n, m, LCC, diam	5	5	3	2	5	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
n, m, LCC, eff-diam	22	24	23	22	23	0	2	9	16	1	3	0	0	0	21	26	29	16	16	14	18	16	1	0	20	0	0	0
n, m, diam	4	3	2	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n, m, eff-diam	22	25	21	22	22	9	2	8	17	2	2	1	0	0	17	23	24	13	16	15	17	19	11	13	17	0	0	0
n, m, k-core	9	9	7	9	10	10	1	2	2	2	1	2	2	1	1	5	16	2	2	1	2	2	1	2	2	1	7	0
n, m, k-core, LCC	11	11	12	12	12	0	4	4	4	2	3	1	0	0	5	10	18	4	5	4	4	2	0	0	5	0	0	0
n, m, k-core, deg	5	5	4	5	5	9	0	0	0	0	0	0	0	0	0	1	14	0	0	0	0	0	0	0	0	0	5	0
n, m, deg	47	49	48	48	48	48	49	49	50	49	48	46	46	48	47	49	48	50	50	50	50	49	48	47	43	50	50	50
n, m, deg, tau	26	28	33	35	38	47	42	37	38	41	44	45	38	37	35	32	29	35	38	40	40	42	43	43	36	42	0	0
n, m, deg, betw	39	39	40	37	32	12	18	36	41	16	16	10	9	9	39	34	24	41	40	42	36	32	29	29	33	3	2	4
n, m, deg, LCC	48	45	47	47	46	0	49	49	47	50	49	42	6	0	48	49	48	49	49	49	49	36	2	0	43	0	0	0
n, m, Katz	47	45	45	46	47	45	13	12	11	12	12	14	16	14	11	13	17	12	12	12	11	11	13	12	13	12	21	4
n, m, Katz, LCC	47	45	46	45	45	0	14	13	12	12	14	15	6	0	13	14	20	12	13	14	12	14	1	0	15	0	0	0
n, m, Katz, deg	46	45	48	45	47	47	14	14	13	13	12	16	17	14	14	15	18	13	13	14	15	13	14	14	14	13	22	6
n, m, PR	48	49	49	49	49	48	49	50	49	50	49	47	47	47	48	50	50	50	50	50	50	49	49	48	46	49	49	50
n, m, PR, LCC	48	47	47	46	46	0	49	49	50	48	49	39	5	0	49	49	48	49	48	48	50	34	1	0	45	0	0	0
n, m, PR, deg	47	49	47	48	48	47	48	49	49	49	48	46	44	47	47	49	49	50	50	50	50	48	47	47	44	50	48	50

Figure 3.7: Blasius Framework classification results based on just mean feature values

Features	1ccu	2ccu	3ccu	4ccu	5ccu	CL-c	1-2-34	1-23	1-234	12-34	2m	3m	4m	5m	1cu	2cu	3cu	2d	3d	4d	5d	6d	7d	hyper	CL	BA	ER	real
betw, close, diam	2	1	0	2	1	1	27	43	48	29	27	20	18	18	32	21	13	49	45	38	29	28	30	32	13	7	15	0
betw, close, eff-diam	13	20	18	18	15	11	25	42	47	25	25	19	17	18	39	29	25	47	42	38	29	38	36	31	12	2	15	17
LCC, tau, diam	4	3	2	3	3	0	31	43	45	33	32	26	10	0	33	22	14	47	44	40	26	2	0	37	0	0	0	0
LCC, tau, eff-diam	25	23	25	24	27	0	32	45	47	31	33	25	8	0	43	38	33	46	42	39	33	2	0	41	0	0	0	19
n, m	48	48	49	48	49	48	49	49	49	50	49	48	46	47	49	49	49	50	50	50	49	49	48	46	50	50	50	50
n, m, betw	29	32	26	27	25	5	4	31	44	5	4	1	1	1	40	21	13	46	37	25	23	23	25	25	0	0	0	39
n, m, betw, close, diam	0	0	0	0	0	0	6	33	45	5	6	4	3	2	30	18	9	41	39	27	19	23	21	25	1	2	2	0
n, m, betw, close, eff-diam	14	19	14	14	15	7	5	30	45	7	5	4	3	3	36	20	13	46	37	25	21	28	27	22	2	1	1	16
n, m, close	3	3	2	2	3	2	31	48	50	30	29	26	26	25	33	4	2	45	47	48	44	31	28	39	17	1	1	2
n, m, close, LCC	4	4	4	3	4	0	31	48	49	29	31	23	5	0	35	4	5	44	48	49	31	2	0	39	0	0	0	4
n, m, close, deg	0	0	0	0	0	0	26	48	49	27	28	24	24	21	30	0	0	40	45	48	42	29	23	35	11	0	0	0
n, m, LCC	23	17	17	19	15	5	13	38	49	13	10	8	6	4	43	29	22	44	44	25	22	35	34	25	3	3	3	31
n, m, LCC, betw	22	25	23	25	19	0	20	44	49	19	17	14	5	0	45	33	22	48	38	25	11	2	0	32	0	0	0	36
n, m, LCC, tau	24	22	20	22	15	8	16	42	48	17	13	11	9	6	41	27	26	46	45	28	24	37	37	29	6	4	6	36
n, m, LCC, tau, diam	47	47	47	45	46	0	47	49	46	48	48	37	6	0	49	48	48	48	47	49	32	1	0	46	0	0	0	49
n, m, LCC, tau, eff-diam	24	29	25	27	24	0	6	33	47	5	5	5	4	0	41	20	10	48	37	25	10	1	0	20	0	0	0	43
n, m, LCC, diam	38	38	44	41	44	0	41	48	49	41	39	36	8	0	46	44	42	47	43	42	30	2	0	46	0	0	0	35
n, m, LCC, eff-diam	2	0	0	1	0	0	29	44	47	32	30	24	6	0	32	18	12	47	42	36	20	0	0	37	0	0	0	0
n, m, diam	20	23	24	24	25	0	28	42	47	26	27	21	2	0	39	27	23	47	43	33	19	2	0	38	0	0	0	15
n, m, eff-diam	1	0	0	1	0	0	31	44	47	35	33	25	6	0	32	20	12	49	46	40	20	0	0	37	0	0	0	0
n, m, k-core	17	21	21	19	19	0	26	41	47	27	26	17	1	0	35	25	14	46	44	37	16	1	0	31	0	0	0	16
n, m, k-core, LCC	47	48	48	48	47	46	48	49	49	49	49	46	47	46	48	47	49	49	50	50	49	47	47	43	49	50	49	50
n, m, k-core, deg	39	38	43	46	45	36	42	48	49	40	42	42	39	38	48	44	41	47	44	42	38	41	40	45	42	0	0	35
n, m, deg	29	33	30	28	25	6	5	36	45	4	5	3	2	2	43	23	15	46	37	27	21	30	26	28	1	1	1	41
n, m, deg, tau	47	45	44	45	45	0	48	49	48	48	48	41	7	0	48	49	48	50	49	49	34	2	0	43	0	0	0	49
n, m, deg, betw	1	0	0	1	0	1	32	44	48	34	32	26	24	22	33	20	10	50	48	40	30	35	41	37	20	19	20	0
n, m, deg, LCC	15	17	18	17	17	17	17	39	46	18	19	13	10	9	35	24	15	47	43	37	26	37	42	31	7	0	22	13
n, m, Katz	8	7	7	9	10	13	49	49	50	49	49	47	46	43	46	34	19	50	49	49	49	47	49	43	50	1	2	12
n, m, Katz, LCC	9	9	10	11	10	0	48	48	48	48	48	39	6	0	46	37	23	50	49	49	33	1	0	45	0	0	0	12
n, m, Katz, deg	7	8	8	10	9	13	48	50	50	47	47	46	45	45	47	38	22	50	50	49	48	47	47	44	49	1	3	13
n, m, PR	48	49	49	49	47	48	49	49	49	50	49	47	46	47	49	49	50	50	50	50	49	49	49	46	50	50	50	50
n, m, PR, LCC	49	47	47	46	46	0	48	48	47	49	48	39	6	0	49	48	49	49	48	49	34	2	0	45	0	0	0	49
n, m, PR, deg	48	48	48	48	47	47	49	49	49	50	48	46	46	46	48	49	48	50	49	50	49	47	47	44	49	50	50	50

Figure 3.8: Blasius Framework classification results based on just mean feature values - "control" here by classification against the 1d max norm torus GIRG. I.e. all GGMs are still fit to the real graph (so for GIRGs mostly having matching mean node degre and mean LCC), but now the SVM is classifying each GGM against the 1d max norm torus GIRG instead of the real graph.

Chapter 4

Diffusion Maps

Contents

4.1 Introduction

The GGM comparison framework detailed in chapter 3 was designed to assess the similarity of two distributions of graphs, real-world and fake generated, based on higher level features. The fake GGM generates are only loosely fit on their real counterparts, such that the graphs have no node-node or edge-edge correspondences. In the non-copy-weight GIRG case, the n nodes of the generated graph have their weights iid power law sampled $w_1, ..., w_n \sim \text{powerlaw}(\tau)$, and their locations iid uniformly sampled $\boldsymbol{x}_1, ..., \boldsymbol{x}_n \sim U(\chi)$. Hence the features used for comparison to the real graphs, like mean (or median / standard deviation) node degree, node closeness centrality, or effective diameter etc. are necessarily node permutation invariant.

An alternative framework is to try to fit a GGM model to a graph so as to compare on an edgewise and likelihood basis. There is little hope for Chung-Lu, Erdos-Renyi and Barabasi-Albert to achieve this. However the GIRG GGM has more parameters, so could fit not just $\hat{\alpha}$ to match the local clustering coefficient, $\hat{c}$ to match the number of edges, and $\hat{\tau}$ to match the degree distribution tail, but to further actually try and infer individual node weights $w_u \in \mathbb{R}^+$, and positions $\boldsymbol{x}_u \in \chi$.

We saw this already to some extent with the copy-weight GIRGs - where $\hat{w}_u = d_u$ is fit from the observed real graph node degrees. The $\boldsymbol{x}_u$ are however much harder to fit; for starters any maximum likelihood fit $(\hat{\boldsymbol{x}}_u)_{u\in V}$ would be rotation, reflection, and translation invariant (isometries). Finding any one maximum likelihood fit is impractical for all but the smallest graphs.

In this chapter we explore the method of *diffusion maps* for computationally efficiently fitting an initial good guess of the $(\boldsymbol{x}_u)_{u\in V}$ positions. [García-Pérez et al., 2019] similarly used diffusion maps to fit a first estimate of node locations for HRGs - this seems to be key to part of their claim of their algorithm being the best HRG embedder with still reasonable $O(n^2)$ runtime. We elucidate more on the applicability/use of diffusion maps, provide some post processing techniques to improve the quality of node estimates, and propose the eigenvalues of the diffusion map as a method for determining an appropriate dimensionality d for the GIRG model.

4.2 Diffusion Maps Theory

Diffusion Maps [Coifman and Lafon, 2006] are a technique originally intended to find a lower dimensional representation of some vector points taken from a higher dimensional space. For this to work, the points must actually conform to a lower dimensional manifold $\mathcal{M}$, e.g. points on a 3d sphere $x_1^2 + x_2^2 + x_3^2 = 1$, but embedded in $\mathbb{R}^5$. Or a "swiss roll": an inwardly spiralling 2d manifold in 3d space. The method involves first converting points to a (weighted) adjacency graph W via a distance kernel, and then decomposing a diffusion process on this graph to produce a lower dimensional vector representation of the original points.

This method is hence suitable for our purposes of $\boldsymbol{x}_u$ inference, as though we don't have a set of high dimensional points, we do have a simple graph $G = (V, E)$ that, like the distance kernel produced graph of diffusion maps, is assumed to have been produced by a geometry influenced process (e.g. generated from a GIRG).

Relation to node location likelihood maximisation [Belkin and Niyogi, 2001] presents the method of Laplacian eigenmaps which is an interpretation of diffusion maps, showing that it optimises for the objective

$$\min_X \sum_{u,v} W_{uv} \|\boldsymbol{x}_u - \boldsymbol{x}_v\|^2 \quad \text{subject to } X^T D X = I;\ X^T D\mathbf{1} = \mathbf{0} \tag{4.1}$$

where $X \in \mathbb{R}^{n\times d}$ is a matrix whose rows $\boldsymbol{x}_u \in \mathbb{R}^d$ are the lower dimensional node representations, and D is the diagonal degrees matrix of W. I.e. if nodes u, v are connected directly by $W_{uv} = 1$, we want that their geometric locations $\boldsymbol{x}_u, \boldsymbol{x}_v$ are close by. For fitting GIRG locations we would actually like to solve the objective of likelihood maximisation

$$\max_{(\boldsymbol{x}_u\in\mathbb{T}^d)_{u\in V}} \sum_{u,v} W_{uv} \log p_{uv} + \overline{W}_{uv} \log \overline{p}_{uv} \tag{4.2}$$

$$\text{where } p_{uv} = \min\left\{1, c\left(\frac{w_u w_v / \sum_j w_j}{\|\boldsymbol{x}_u - \boldsymbol{x}_v\|^d}\right)^\alpha\right\} \quad \text{and } \overline{a} = 1 - a \tag{4.3}$$

Although the two objectives are different, we hope that the efficiently solvable diffusion map provides a good first approximation of the maximum likelihood locations.

4.2.1 Random walk formulation of diffusion maps

The random walk formulation of diffusion maps helps to give us intuition for how to effectively use the method with GIRG generated graphs.

The idea is to analyse the diffusion process of randomly walking on the edges of the graph, characterising the probability cloud starting from one node in the graph as a sum of decreasingly important orthogonal contributions - coming from the eigenvectors of decreasing eigenvalues of a diagonalisable matrix. The top d contributions can then be used as a coordinate system to describe each point in the graph. By taking a large timestep diffusion cloud, the general relative location of the initial point is the main signal. The hope is that if connections (probabilistically) follow a geometry of d-dimensions, then the diffusion map coordinate system will capture / align with this real geometry.

The relevant quantities of the markovian random walk $u_t : t \in \mathbb{N}$ are defined

$$W_{uv} = \begin{cases} 1 & u \sim v \\ 0 & u \nsim v \end{cases} \qquad \text{node adjacency matrix}$$

$$M_{uv} := P(u_{t+1} = v | u_t = u) = \frac{W_{uv}}{\deg(u)} \qquad u \to v \text{ transition probability}$$

$$M = D^{-1}W \qquad \text{transition matrix}$$

$$D_{uu} = \sum_{v \in V} W_{uv} \qquad \text{diagonal degree matrix}$$

$$S = D^{-1/2}WD^{-1/2} \qquad \text{symmetric matrix}$$

$$= V\Lambda V^T \qquad \text{diagonalisation into orthonormal e-vectors}$$

$$\Phi = D^{-1/2}V = [\boldsymbol{\phi_1}, \boldsymbol{\phi_2}, ..., \boldsymbol{\phi_n}] \qquad \Psi = D^{1/2}V = [\boldsymbol{\psi_1}, \boldsymbol{\psi_2}, ..., \boldsymbol{\psi_n}]$$

$$\Phi^T\Psi = \Psi^T\Phi = I_{n \times n} \qquad \text{due to orthonormality of } V$$

We use the diagonalisation of S to write M as

$$M = D^{-1/2}SD^{1/2} = \Phi\Lambda\Psi^T \qquad \text{diffusion map representation}$$
$$= \sum_{k=1}^{n} \lambda_k \boldsymbol{\phi}_k \boldsymbol{\psi}_k^T$$

The diagonalisation of sparse [1] symmetric matrix S can be done quite efficiently: we use `scipy.sparse.linalg.eigsh` to find the top $d+1$ eigenvalues.

The biorthonormality $\langle \boldsymbol{\phi}_i, \boldsymbol{\psi}_j \rangle = \delta_{ij}$ means that $M\boldsymbol{\phi}_i = \lambda_i \boldsymbol{\phi}_i$ and $M^T\boldsymbol{\psi}_i = \lambda_i \boldsymbol{\psi}_i$. For a diffusion map representation of nodes, we order eigenvalues $\lambda_1 \geq \lambda_2 \geq ....$ Notably the transition matrix satisfies

$$M\mathbf{1} = \mathbf{1} \quad \text{since} \quad \sum_j \frac{W_{ij}}{\deg(i)} = 1 \tag{4.4}$$

$$\implies \lambda_1 = 1 \text{ and } \boldsymbol{\phi}_1 = l\mathbf{1} \text{ for some scalar } l \tag{4.5}$$

as it can be shown that $\lambda_2 < 1$ if the graph is connected, with equality only if it is disconnected.

[1] nodes in GIRGs have $O(1)$ average degree; nodes in socfb Facebook graphs generally have average degree less than 90 - hence they are relatively sparse.

The diffusion map representation of a node is then

$$
\begin{aligned}
e_u^T M &= \textstyle\sum_{k=1}^{n} \phi_u(k)\lambda_k^t \boldsymbol{\psi}_k && \text{diff map of node i after t steps}\\
\boldsymbol{\varphi}_t(u) &:= (\phi_2(u)\lambda_2^t, \ldots \phi_{d+1}(u)\lambda_{d+1}^t) && \text{d-trunc diff map representation}\\
D_t(v_i, v_j)^2 &:= \textstyle\sum_k \frac{1}{d_k}(M_{ik}^t - M_{jk}^t)^2 && \textit{diffusion distance}\\
D_t(v_i, v_j)^2 &= \left\|\boldsymbol{\varphi}_t(v_i) - \boldsymbol{\varphi}_t(v_j)\right\|^2 && \text{provable equality}
\end{aligned}
$$

The truncated diffusion map summarises the distinguishing features of a node's random walk cloud after t steps, removing the tail of decaying contributions which are scaled by smaller $\lambda_{i>d+1}$, Since $\boldsymbol{\phi}_1 = l\mathbf{1}$, the first coordinate is dropped as it's useless for distinguishing nodes. This corresponds to the fact that the diffusion cloud starting at any node converges as $t \to \infty$ to the same *stationary distribution* $\boldsymbol{\pi}_i = \frac{d_i}{\sum_j d_j}$, the unique normalised solution to $M^T\boldsymbol{\pi} = \boldsymbol{\pi}$.

4.2.2 Improving diffusion map geometic distance preservation

We ideally want the diffusion map embedding of a GIRG generated graph to have similar distances to the original geometry: $\|\boldsymbol{\varphi}(u) - \boldsymbol{\varphi}(v)\| \approxeq \|\boldsymbol{x}_u - \boldsymbol{x}_v\|$, as interpoint distances determine edge probabilities in our GIRG.

Long story short, in the conventional diffusion map setting of manifold $\mathcal{M}$ in high dim space $\to$ lower dim embedding, the diffusion map is achieving a discrete approximation of the Laplace-Beltrami operator on the whole continuous manifold - see [Singer, 2006]. This operator defines a continuous diffusion distance $D_t(x,y) = \|\boldsymbol{\varphi}_t(\boldsymbol{x}) - \boldsymbol{\varphi}_t(\boldsymbol{y})\|$ between points $x, y \in \mathcal{M}$, which is approximately equal to the geodesic distance $d_g(x,y)$ on the manifold (up to a scale factor), when $d_g(x,y)^2 \ll t \ll 1$, as shown in the background section of [Berry and Harlim, 2018]. Hence if we can achieve a good discrete approximation to the continuous diffusion process, we can hope to get embeddings nearly isometric to the true geometric locations.

With GIRGs, our "manifold" is the original geometric space χ, e.g. a torus or a cube. Each node $u \in V$ has a true location $\boldsymbol{x}_u \in \chi$, and instead of having distances $d(u,v)^2$ in some higher dimensional space we just observe the presence of edges/non-edges, $u \sim v$ or $u \nsim v$, which give a signal as to whether $\|\boldsymbol{x}_u - \boldsymbol{x}_v\|$ is large (if $u \nsim v$) or small (if $u \sim v$).

We highlight three different approaches in implementing a diffusion map embedding of an input simple unweighted graph (suspected to be a GIRG) $G = (V, E)$. G has adjacency matrix $A_{ij} = \begin{cases} 1 & \text{if } u \sim v \\ 0 & \text{if } u \nsim v \end{cases}$, and the differences between the approaches can be encapsulated in the choice of the weighted adjacency matrix W_{ij} seen in section 4.2.1 used to define the diffusion process.

Naive approach The naive approach is just to use $W = A$. This is a valid methodology akin to taking edge weights 1 or 0 between points $\boldsymbol{z}_i, \boldsymbol{z}_j$ in high dim space if they fall within some $\varepsilon > 0$ distance of each other; this is the "Simple-Minded" variant proposed in [Belkin and Niyogi, 2001]. Furthermore if we wanted to make minimal assumptions on the generative process that produced G, beyond that edges were formed by some geometric closeness criterion, then this is probably the best approach.

4.2.2.1 Expected distance approach

The approach used by [García-Pérez et al., 2019] to estimate node locations for the Hyperbolic Random Graph model uses the second "Heat Kernel" edge weighting variant of [Belkin and Niyogi, 2001]. This is specified as taking

$$W_{ij} = e^{-\left\|z_i - z_j\right\|^2 / T} \quad \text{for some parameter } T > 0 \tag{4.6}$$

as adjacency matrix weights between vectors z_i, z_j in the high dim space. This is designed to optimise the discrete approximation of the continuous heat diffusion on the manifold $\mathcal{M}$.

The idea of [García-Pérez et al., 2019] is to substitute for $\left\|z_i - z_j\right\|^2$ the expected distance between nodes u, v in the HRG - we can do the same for a GIRG: $\mathbb{E}[\|\boldsymbol{x}_u - \boldsymbol{x}_v\|^2]$. This is our best distance guess given our assumption that G is GIRG generated.

For a connected pair of nodes $u \sim v$, with weights w_u, w_v,

$$E[r_{uv}^2 | w_u, w_v, u \sim v] = \Theta\left[\left(\frac{w_u w_v}{W}\right)^{\eta}\right] \tag{4.7}$$

where $\eta = \min(\frac{2}{d}, \alpha - 1)$, and we focus on the largest scaling term for simplicity. We can use the node weight estimates of $\hat{w}_u = d_u$ to get distance estimates $\hat{r}_{uv}^2 = \left(\frac{w_u w_v}{W}\right)^{\min(2/d, \alpha-1)}$. Finally edge weights are $W_{uv} = e^{-\hat{r}_{uv}^2/T}$, i.e. decaying with predicted distance.

4.2.2.2 Homogeneous random walk approach

The final approach we consider acts similarly to the expected distance approach, downweighting edges between nodes u, v which have high degree (estimated weight) - intuitively this edge is likely to be longer in the χ-space and hence less reflecting of the geometry. We aim to make the random walk on nodes of the graph G directly similar to a (geometric) random walk on the χ-space - i.e. making it heat diffusion / brownian motion like.

Again we take d_u as the estimated weight $\hat{w}_u$ of node u. We can modify the naive random walk M to prioritize neighbours v with lower degree d_v via

$$M_{uv} \leftarrow \frac{M_{uv} d_v^{-\gamma}}{\sum_{v'} M_{uv'} d_{v'}^{-\gamma}} \tag{4.8}$$

$$\text{i.e.} \quad M \leftarrow D' M D^{-\gamma} \quad \text{(diagonal } D' \text{ normalises probabilities)} \tag{4.9}$$

for some parameter $\gamma \geq 0$. This modification of transition probabilities is viewed more properly as modifying original edge weights such that $W_{uv} = k(d_u)k(d_v)$, where $k(d_u) = d_u^{-\gamma}$.

Intuitively, picking $\gamma = 1$ seems like a good start - in powerlaw weighted GIRGs, it perfectly counterbalances the tendency for a node's neighbours to have a shifted weight distribution. In the naive approach where $W_{uv} = 1$ for $u \sim v$ model, the next state of the random walk starting at u uniformly randomly samples a neighbour $v \sim u$ which then has weight distributed as a shifted power law $w_v \sim$

powerlaw$(\tau-1)$; our modified random walk has $w_v \sim$ powerlaw(τ), matching the original node weight distribution.

Furthermore with no assumptions on the weight distribution, $\gamma = 1$ also means that the random walk's stationary distribution over the nodes is more like a constant $\propto \mathbf{1}$ distribution rather than the known $\pi_u = \frac{d_u}{\sum_v d_v} \propto d_u$ distribution. We can see this by using the fact that the modified node degrees is now $\widetilde{d}_u = \sum_{v\sim u} d_u^{-1} d_v^{-1} = d_u^{-1} \sum_{v\sim u} d_v^{-1}$. Informally, for graphs where most nodes have a decent number of neighbours spread across lower and higher degree nodes, which includes our powerlaw degree sequenced GIRGs, then $\sum_{v\sim u} d_v^{-1} = \Theta(d_u)$, and so $\widetilde{\pi}_u \propto d_u^{-1}\Theta(d_u) = \Theta(1)$.

A homogeneous final stationary distribution is the outcome of heat diffusion on a manifold, so is perhaps desirable.

4.2.2.3 Bounded diffusion hop distances

Another angle to view how our three different W weighting approaches might produce diffusion map point embeddings that better preserve original geometric distances is the expected diffusion hop distance. In order to achieve distance preserving diffusion map point inference, we want the diffusion process to closely match brownian motion on the original manifold, in our case the geometric space $\chi = [0,1]^d$. Consider one random walk step starting at node u and choosing a neighbour $v \sim u$ to transition to. While the direction in which we go is roughly random (e.g. for $\chi = [0,1]^2$ a 2d square, any neighbour $v \sim u$ is relatively positioned with $\boldsymbol{x}_v - \boldsymbol{x}_u$ having an angle $\theta \in [0, 2\pi)$ uniformly randomly), the distance we travel is not uniformly distributed. For our random walk to approximate brownian motion we want the "hop distance" $r_{u_t u_{t+1}}$ to have *finite variance*.

Rescaled geometric space For $\chi = [0,1]^d$ hops are of course bounded to be finite; we actually have to view the GIRG in a new lens for this argument to make sense. Without going too much into detail, a GIRG can be equivalently formulated by replacing $\chi \leftarrow [0, n^{1/d}]^d$ and redefining $p_{uv} = \min\left\{1, c\left(\frac{w_u w_v}{r_{uv}^d}\right)^\alpha\right\}$ (i.e, removing the $1/W$ scaling on the node weight product [2]). This means that as we increase $n \to \infty$, the uniformly distributed node points don't get more tightly packed into the $[0,1]^d$ cube, instead they're always spaced with mean density of occupying 1 unit of volume per point.

We wish to calculate the expected squared hop distance in the rescaled χ-space, $E[r_{uv}^2 | w_u, u \to v]$, where $u \to v$ is the event that we transition from node u to v in one step of the random walk. First ingredients we need:

$E[r_{uv}^2 \mid w_u, w_v, u \sim v] = \Theta(w_v^\eta)$	where $\eta = \min(\frac{2}{d}, \alpha - 1)$, dropping factors of w_u
$p(w_v \mid w_u, u \sim v) \propto w_v^{1-\tau}$	shifted powerlaw weights
$u \to v = u \to v \cap u \sim v$	equivalent events

[2] note that whp $W = \Theta(n)$ for $\tau > 2$ powerlaw weights, so we've swapped the downscaling of edge probabilities by $1/n$ from the weights to the distances

We also need

$$p(w_v = w|w_u, u \to v, u \sim v) \propto p(u \to v|w_u, w_v = w, u \sim v)\, p(w_v = w|w_u, u \sim v)$$
$$\propto p(u \to v|w_u, w_v = w, u \sim v)\, \Theta(w^{1-\tau})$$

And finally, for each of the differente adjacency matrix weighting schemes,

$$p(u \to v|w_u, w_v = w, u \sim v) \approx\propto \qquad \text{(assuming degrees } d_i \cong w_i\text{)}$$
$$\begin{cases} \frac{1}{w_u} & \text{Naive approach} \\ e^{-\hat{r}_{uv}^2/T} = \exp(-\frac{1}{T}(\frac{w_u w}{W})^\eta) = \exp(-Kw^\eta) & \text{Expected distance} \\ w^{-\gamma} & \text{Homogeneous} \end{cases}$$

Putting this together we have a formula for the expected hop distance in one discrete timestep of the diffusion process

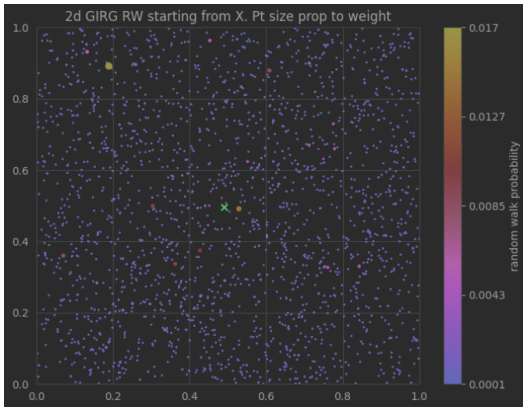
$$E[r_{uv}^2|w_u, u \to v] \tag{4.10}$$
$$= \int_1^\infty E[r_{uv}^2|w_u, u \to v, w_v = w] p(w_v = w|w_u, u \to v) dw \tag{4.11}$$
$$= \int_1^\infty E[r_{uv}^2|w_u, w_v = w, u \sim v] p(w_v = w|w_u, u \to v) dw \tag{4.12}$$
$$= \int_1^\infty \Theta(w^\eta) p(w_v = w|w_u, u \to v) dw \tag{4.13}$$
$$= \int_1^\infty \Theta(w^{1+\eta-\tau}) p(u \to v|w_u, w_v = w, u \sim v) dw \tag{4.14}$$

In the naive approach, similarly to how $\int_1^\infty x^\beta dx$ is bounded if $\beta < -1$, if $1 + \eta - \tau > -1$ then the hop distance is unbounded, i.e. if $\tau < 2 + \min(\frac{2}{d}, \alpha - 1)$. This is eminently possible for $\tau \in (2, 3)$. The expected distance approach has no problems as $\exp(-Kw^{2/d})$ decays swiftly, however in the homogeneous approach taking $\gamma = \min(\frac{2}{d}, \alpha - 1)$ would guarantee bounded hop distances for all $\tau \in (2, 3)$, as $2 - \tau < 1$.

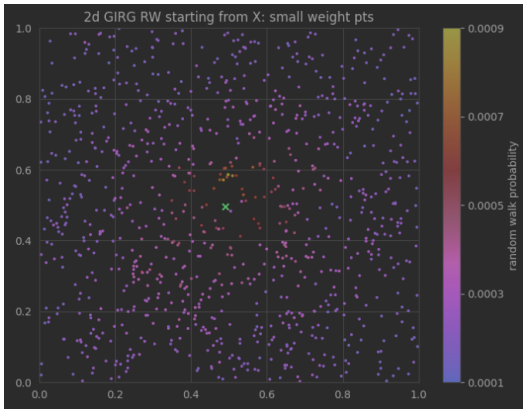
In fig. 4.1 diffusion clouds out of a point are shown for the three different edge weighting approaches. Both non naive versions look more like brownian motion clouds as desired. We additionally benchmarked the approaches by comparing the diffusion map embedding of a fake graph generated from a GIRG with known χ-space embeddings. The homogeneous random walk approach has the best correlation between embeddings and true locations, although with $\gamma = 0.9$ not $\gamma = 1.0$.

4.3 Empirical Results and Post-Processing

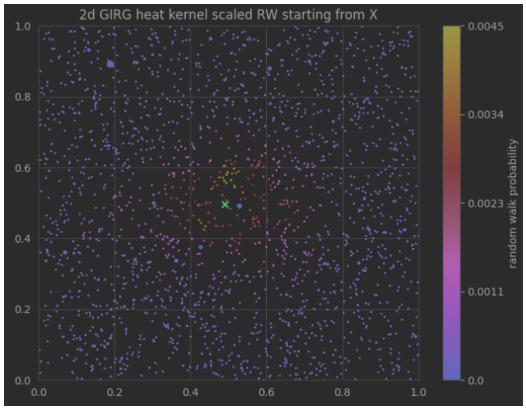
We see in fig. 4.2 some 2d truncated diffusion maps. When the underlying graph was a 2d cube GIRG, the inferred points indeed look square like. From the 1d GIRG we get a curved line shape - i.e. a 1d manifold embedded in 2d, as expected. In testing diffusion maps on max norm torus / cube geometries, replication of node locations was very good. We additionally tested distorted GIRG variants to see if atenuating the signal of some dimensions would monotonically degenerate their diffusion map embedding - this was observed to be the case. We did not test diffusion maps on MCD GIRGs, though this would be interesting. We think in theory that, although the brownian motion would look unusual, no longer



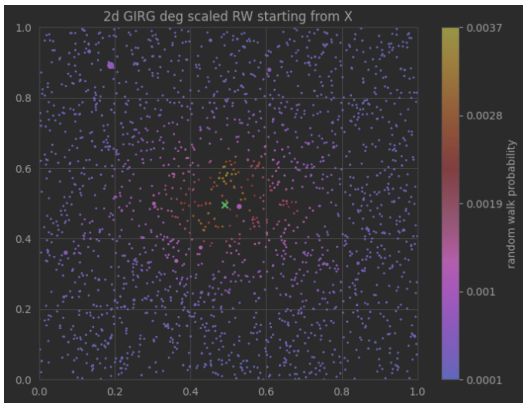
(a) Naive edge weightings: Large weight nodes get disproportionately more probability mass.



(b) Restriction of Naive approach plot to nodes with smaller weights. Probability distribution looks more gaussian



(c) Heat Kernel expected edge distance scaled edge weights $W_{uv} = e^{-\hat{r}_{uv}^2}$. Helps to remove the large weight bias - perhaps too much as large weight nodes are too cold.



(d) Homogeneous approach scaling edge weights by node degree d_i^{-1} also removes the large weight bias, seemingly more evenly

Figure 4.1: $t = 6$ step random walk diffusion cloud on a 2d torus GIRG. Each point is a node shown at its location $x_u \in [0,1]^2$. $u_{t=0}$ is the node marked with the green x. Point colour denotes probability mass in the diffusion cloud.Point radius is proportional to node weight w_u. Three different random walks are shown corresponding to our three different graph edge weighting (transition probability) schemes.

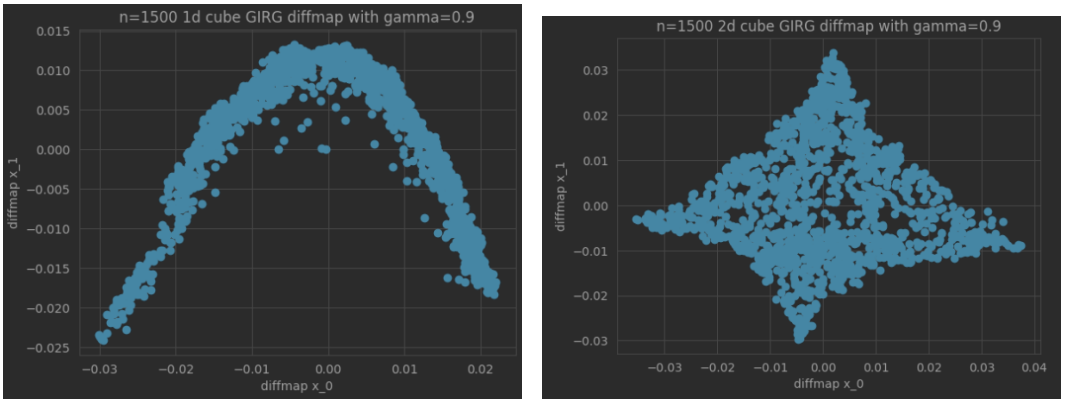
euclidean geometry like, the diffusion map embedding should still be able to recover the original geometry. This would be good to test in future work.

We detail some post processing steps that can be done to improve diffusion map embedding for both GIRG generated graphs, and real graphs (real graphs in particular produce wilder diffusion maps due to their idiosyncratic connectivity patterns).

4.3.1 Inferring Dimension d

If we have a graph G which we know is generated from a d-dimensional GIRG, we can simply extract the d-truncated diffusion map coordinates of each node as an initial estimate for the original geometric location of the node.

However if the true geometric dimension d is unknown, an important first question is how to choose the output dimension d (truncation for the embedding) of the



(a) 1d cube GIRG 2d diff map **(b)** 2d cube GIRG 2d diff map

Figure 4.2: Diffusion map scatter plot of the 2d truncated diffusion maps of (a) graph from a 1d cube GIRG, (b) graph from a 2d cube GIRG. Each plotted point is the inferred location of a single node.

diffusion map.

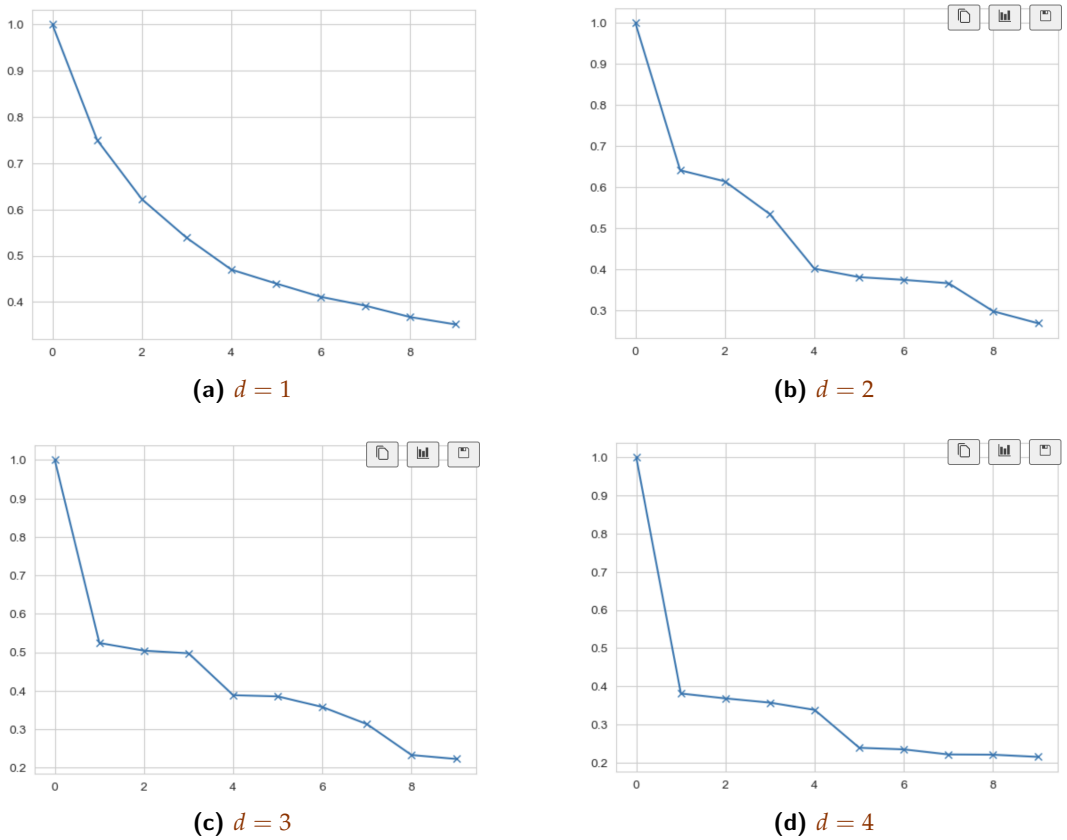
Diffusion maps actually present one way to infer the dimensionality by analysing the ordered sequence of eigenvalues $\lambda_2 < \lambda_3 < ....$ For example in fig. 4.3 we see that for graphs generated from cube GIRGs with dimension $d = 1, 2, 3, 4$, the diffusion map eigenvalue have a clear cutoff point after the first $d + 1$ eigenvalues, with $\lambda_2, ..., \lambda_{d+1}$ being all approximately equal. Hence an eigenvalue cutoff can be used to infer geometric dimensionality of a graph. This is inherently manual/empirical, and so more difficult for real world graphs.

Toroidal GIRGs diffusion map eigenvalues Interestingly Toroidal GIRGs are differentiated from Cube GIRGs in that the diffusion map requires $2d$ coordinates to capture the Torus geometry instead of just d for the cube. The natural diffusion map of a 1D Torus GIRG ends up being a 2d circle about the origin; 2 large eigenvalues λ_2, λ_3 are necessary. This representation of the torus does accurately retain interpoint distances which is what we care for the most, however if you really knew that it was a torus of some dimension, you could project the embedding back into the torus space for a more compact representation. For instance [García-Pérez et al., 2019] map 2d points to an angle of direction away from the origin $\phi \in [0, 2\pi)$ using arctan.

4.3.2 Rescaling/Shifting Diffusion Maps

From section 4.2.2 we know that the diffusion map embeddings should be roughly isometric up to a scale factor - so rescaling and shifting into the unit cube/torus is important to make inter-point distances meaningful/consistent across multiple graphs (even though this could be absorbed into the probability scaling constant c).

The embeddings will be centered around the origin, as all $\lambda_2, \lambda_3, ...$ eigenvalue eigenvectors are orthogonal to $\boldsymbol{\phi}_1 = k\mathbf{1}$: they represent a deviation from the stationary distribution. E.g. in a 1d cube GIRG for a node with $\boldsymbol{x}_u \approxeq 0$ at the extreme geometric left end, its diffusion cloud will need to put more diffusion probability on the left side nodes and less on the right side nodes than the stationary distribution.



(a) $d = 1$ **(b)** $d = 2$

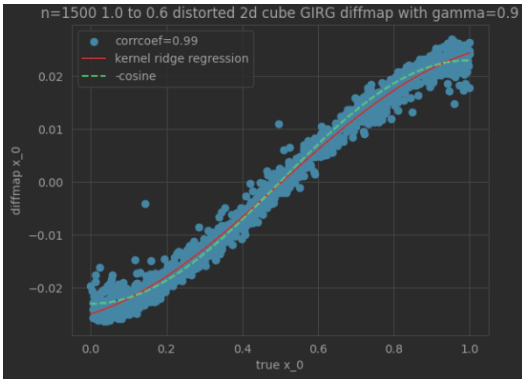
(c) $d = 3$ **(d)** $d = 4$

Figure 4.3: Eigenvalues of the diffusion map laplacian matrix (including $\lambda_1 = 1$) for 4 different $n = 2000, \tau = 2.5, \alpha = 1.3$ cube GIRGs: with dimensions $d = 1, 2, 3, 4$. X-axis is the ith eigenvalue, y-axis is the eigenvalue value.

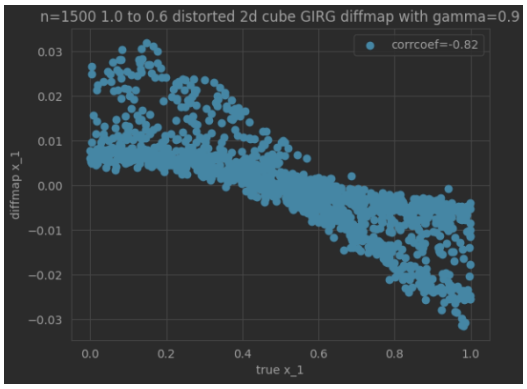
In order to map embeddings into the $[0,1]^d$ torus/cube, the 0 centering can easily be fixed by shifting the points $(\boldsymbol{x}_u)_i \leftarrow (\boldsymbol{x}_u)_i + \min_v(\boldsymbol{x}_v)_i$. If we are certain that the points should be distributed within the unit cube, then we can simply rescale separately along each dimension: $x \leftarrow \frac{x - x_{\min}}{x_{\max} - x_{\min}}$. If furthermore we're certain that the distribution within the unit cube should be relatively uniform, we can perform a coordinatewise uniformify procedure that replaces $(\boldsymbol{x}_u)_i$ with its percentile value compared with other $(\boldsymbol{x}_v)_i$. fig. 4.5 shows the uniformify procedure in action. Real graphs' diffusion maps are often quite non-uniformly located points, with some highly bunched up and others spread out, so uniformify can be quite useful in this case.

Rotated Points One issue with diffusion maps as seen in fig. 4.2b is rotations - the 2d GIRG's points are rotated into a diamond shape. As we only hope for an isometric mapping of the original point locations to the diffusion map embedding, we can't expect the diffusion map embedding to be in the same basis as the original point locations. If we want to fit it back into $[0,1]^d$, we don't care about reflections, and we can translate and rescale, however rotating is more difficult.

That fig. 4.2b looks almost perfectly 45 degree rotated could potentially be explained as the diffusion map trying to maximise diffusion explainability - the long square diagonal has overall more diffusion along it so might be picked up as a major eigenvalue - maybe a small imbalance of more nodes in one corner can cause



(a) First dimension - larger weight scaling in distance function. As the more geometry influencing dimension, it is easier for the diffusion map to fit well, and is picked up as the largest non 1 eigenvalue. The fit is quite close to a cosine wave as shown.



(b) Second (minor) dimension - diffusion map has more trouble fitting this information as it is harder to distinguish from noise. Note a linear relationship means success - negative gradient is irrelevant.

Figure 4.4: 2d distorted cube GIRG. 2 dim truncated diffusion map embeddings shown with one plot for each dimension. Each plotted point is one node from the graph, with X-axis being its true location in the 2d cube for that dimension, and Y-axis being the diffusion map embedding.

this. Even assuming no overall rotation bias, you're going to get a square rotated somewhere between 0 and 45 degrees.

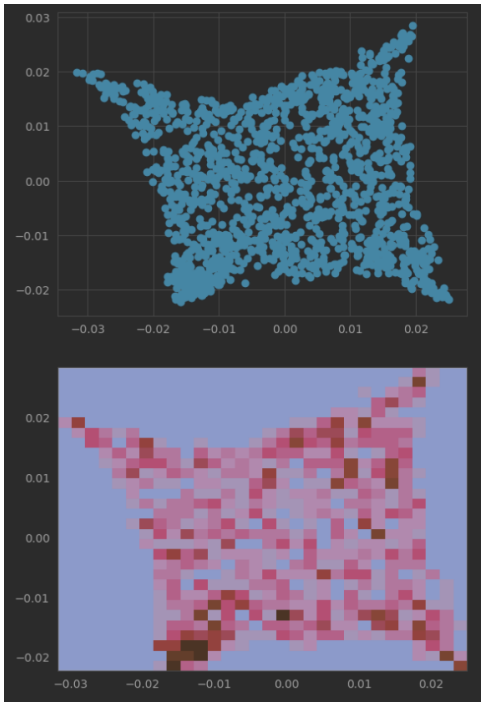
Distorted GIRGs make cuboidal diffusion map embeddings In practice, real graphs never have a perfectly equal balance in geometric dimension importance. We introduced distorted GIRGs in section 2.5 as a possible model for this, where for example the distance function could be the weighted Euclidean norm in 2d:

$$\|x - y\|^2 = a(x_1 - y_1)^2 + b(x_2 - y_2)^2 \tag{4.15}$$

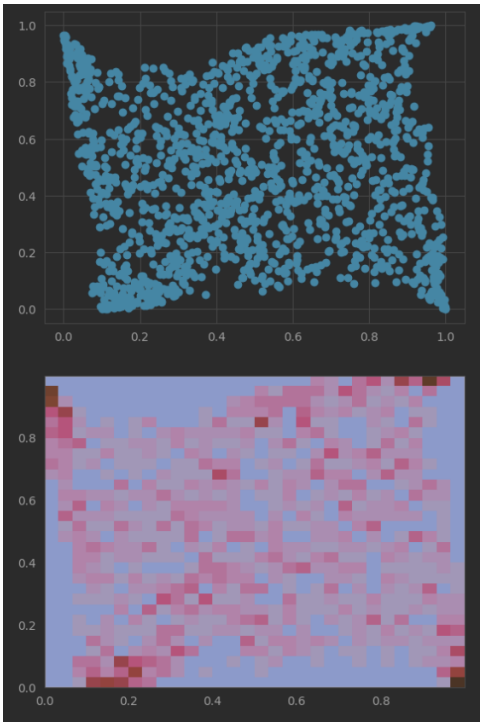
If $a > b$, even just by a little amount, then the first diffusion map coordinate $\varphi_1(u)$ is likely to be very similar to $(\boldsymbol{x}_u)_1$. Only if $a = b$ is there no preference between $(\boldsymbol{x}_u)_1$ and $(\boldsymbol{x}_u)_2$, making a rotation possible. Hence the points $(\varphi_1(u), \varphi_2(u))_{u \in V}$ are likely to end up as a rectangle, not a rotated square; the variation in first coordinate will be greater than in the second.

fig. 4.4a shows the 2d diff map embedding of such a distorted 2d cube GIRG. Note that the major coordinate is much easier to fit well, whereas the minor one has higher error (NB the diff map embedding of x_2 is negatively scaling with the true value which is fine - we can never guarantee same signs). We also see that the x_1 diffusion map embedding does not have quite a linear relationship with the real locations - it looks more like a cosin wave like relationship - this actually makes sense as eigenfunctions of the laplacian operator on a cube look like sin waves.

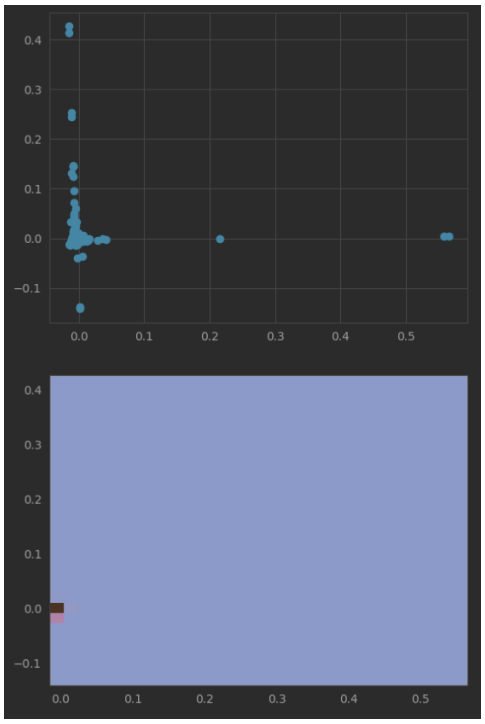
In the light of potentially non equal coordinates, the relative $\lambda_2 > \lambda_3 > ...$ scaling can be seen as a feature not a bug, if the hypothesis space of generative graph models is to be expanded to cuboid (non-cube) GIRGs. In this case all coordinates of the diffusion map don't have to be all (non-homogeneously) rescaled to the range $[0, 1]$ and can rather be homogeneously rescaled to retain their relative size ratios. This sheds new light on the horizontalness of the $\lambda_2, \lambda_3, \lambda_4, \lambda_5$ line segment in fig. 4.3d as a testament to the underlying cubeness (non cuboid) of the GIRG that generated the graph.



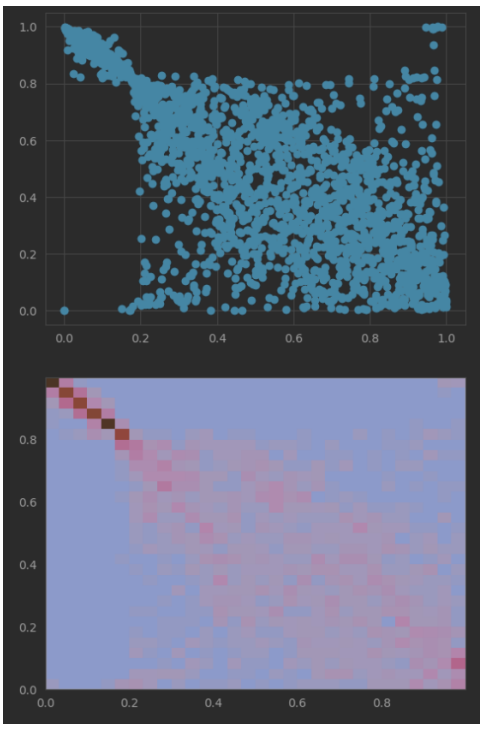
(a) 2d cube GIRG diff map



(b) 2d cube GIRG diff map with uniformify rescaling



(c) socfb-Amherst41 diff map



(d) socfb-Amherst41 diff map with uniformify rescaling

Figure 4.5: 2d truncated diffusion maps plotted - each point is the embedding of a node in the graph. (a) and (b) are on a 2d GIRG, (c) and (d) on a real-world social network graph. The affect of the uniformify point rescaling procedure is shown, in (b) and (d).

Restricted Rescaling This simple post processing method aims to map points into a reasonable geometric space without strongly biasing towards a uniformly distributed prior like the uniformify procedure. Empirically, the diffusion map coordinates of real-world facebook graphs often have $\geq 90\%$ of the nodes concentrated in a small parcel, with only a few nodes having extremely far out locations. This is due to most nodes being very well interconnected, and some being much more torturously linked to the central hub - hence ending up with great diffusion distance separation. This defeats the simple rescaling method of $x \leftarrow \frac{x - x_{\min}}{x_{\max} - x_{\min}}$ as most nodes will end up very tightly packed. However if we hit all the points with the uniformify procedure hammer, we might lose some of the subtleties of the geometry picked up by the diffusion map within the highly connected central parcel.

Instead we only rescale the central nodes: those whose joint coordinate-wise percentiles lie in $[5\%, 95\%]^d$; they are linearly scaled to the $[0.05, 0.95]^d$ cube. Finally the outlying nodes are percentile rescaled (non-linearly) to the upper/lower cube margins. This method is shown in comparison for a few graphs in fig. 5.6

Chapter 5

Likelihood Point Estimation

Contents

5.1 Introduction

Diffusion maps in chapter 4 provide a good initial point estimate for fitting GIRG node locations to a real graph. However what we'd truly like to maximise is the actual data likelihood $p(G|\theta)$, where G is the real graph and θ are all the GIRG parameters like c, α, node weights and node locations. This is infeasible to compute exactly, but there have been many approximate methods developed in the literature for fitting HRGs. To our knowledge this is the first time that a point fitting attempt has been done for higher ($d = 1, 2, 3$) dimensional GIRGs.

In section 5.2 we first tried a *Markov Chain Monte Carlo (MCMC)* approach to not just improve $p(G|\theta)$, but also provide in theory an estimate of the posterior distribution of $p(\theta|G)$. Our use of the *Metropolis-Hastings algorithm* turns out post-hoc to be similar (but less well implemented and analysed) to [Boguná et al., 2010]'s HRG fitting algorithm. We later shifted in section 5.3 to a *direct ordered likelihood optimisation* with rounds of point location updates in order from highest to lowest degree node, inspired by [García-Pérez et al., 2019].

Though our methods are sadly not extremely well thought out or optimal, we still get reasonable results suggesting a good level of fit of the GIRG model already in just 1 dimension to the Facebook social networks, and evidence that adding more dimensions does not improve the fit much more.

5.2 MCMC Formulation

MCMC is a method in the bayesian framework of a parametric generative model. Datapoints z are generated by first sampling $\theta \sim p(\theta)$ from a prior, and then generating from the model $z \sim p(z|\theta)$. In our case of fitting the GIRG GGM, z is a real graph instance G, parameter $\theta = (\alpha, c, (w_u)_{u\in V}, (x_u)_{u\in V})$, and we focus in particular on the node locations $(x_u)_{u\in V}$.

The posterior likelihood $p(\theta|z) = \frac{p(z|\theta)p(\theta)}{p(z)}$ is infeasible to compute due to the normalising factor $p(z)$. MCMC instead uses the non-normalised $Q(\theta) = p(z|\theta)p(\theta)$ which can be evaluated, to set up a *Markov Chain (MC)* with states $\theta \in \Theta$, and transition probabilities derived from $Q(\theta)$. In particular we will use a **Metropolis-Hastings** style MC. The two basic components are a *proposal distribution* to propse a new MC state $\theta' \sim p_{\text{prop}}(\theta'|\theta)$, and an *acceptance probability* $p_{\text{acc}}(\theta'|\theta)$ to decide whether to accept θ'. It might take a few rounds of proposal and rejection to produce one actual MC state transition. This defines a random walk on the MC state space that converges in the limit to the posterior distribution $\theta \sim p(\theta|z)$. If the jth step of the random walk yields θ^j, given sufficient burn in and spacing, we can sample $\theta^{j_1}, \theta^{j_2}, ...$ from the posterior. We will be content with just one posterior sampled $\hat{\theta}$ (NB not a maximum likelihood estimator), and use this to evaluate our overall GIRG model fit to the real graph.

We transition $\theta \to \theta'$ with a *Gibbs sampling* approach. This means breaking down θ into subcomponents θ_i, randomly choosing one to propose a new state for, i.e. $\theta' = (\theta'_i, \theta_{j:j\neq i})$, and using the marginal $Q_i(\theta_i)$ instead of $Q(\theta)$ in order to compute the transition probabilities. In our case $Q(\theta) = p(G|(\boldsymbol{x}_u)_{u\in V})p((\boldsymbol{x}_u)_{u\in V})$, so we can ignore the uniform prior $p((\boldsymbol{x}_u)_{u\in V})$ as it's constant for all node locations. Then we have

$$Q(\theta) \propto \prod_{(u,v)\in\binom{V}{2}} p_{uv}|w_u, w_v, x_u, x_v, \alpha, c, G \tag{5.1}$$

$$\text{marginalised as} \quad Q_u(\theta_u) = \prod_{v\neq u} p_{uv}|... \tag{5.2}$$

Burn in time of the MC can be quite long. With the GIRG model, the natural initialisation for $\boldsymbol{x}_u$ would be to follow the prior $\boldsymbol{x}_u \sim U([0,1]^d)$. By using a diffusion map embedding initialisation this can be greatly reduced.

The proposal distribution should be designed to maximise chances of acceptance. It seemed reasonable to randomly propose one of a small local perturbation, or a random jump to anywhere in the cube A random uniform jump is useful to try and find a completely new location for $\boldsymbol{x}_u$ that could still be nicely near to u's neighbours and far from its non-neighbours - in fact another good proposal would be to randomly choose a neighbour $v \sim u$, and move $\boldsymbol{x}_u$ to a random offset of $\boldsymbol{x}_v$, although this would unfortunately make p_{prop} no longer symmetric. A small perturbation $\boldsymbol{x}'_u = \boldsymbol{x}_u + \varepsilon$ is also good as, assuming that $\boldsymbol{x}_u$ has high likelihood, quite likely a nearby location will have an even higher one.

Acceptance probability as defined by the Metropolis-Hasting's algorithm is

$$p_{\text{acc}}(\boldsymbol{x}'_u|\boldsymbol{x}_u) = \min\left(1,\ \frac{p_{\text{prop}}(\boldsymbol{x}_u|\boldsymbol{x}'_u)p(G|\boldsymbol{x}'_u)p(\boldsymbol{x}'_u)}{p_{\text{prop}}(\boldsymbol{x}'_u|\boldsymbol{x}_u)p(G|\boldsymbol{x}_u)p(\boldsymbol{x}_u)}\right) \tag{5.3}$$

$$= \min\left(1,\ \frac{p(G|\boldsymbol{x}'_u)}{p(G|\boldsymbol{x}_u)}\right) \tag{5.4}$$

$$\text{as } p_{\text{prop}} \text{ is symmetric and } p(\boldsymbol{x}_u) = 1\ \forall \boldsymbol{x}_u \tag{5.5}$$

5.2.1 MCMC likelihood comparison of GIRG vs CL fit to real graph

How well do the converged MCMC points do at replicating the real graph? As suggested in the classification comparison framework, the Chung-Lu model is a good non-geometric null hypothesis to compare against - i.e. we hope that the MCMC fit GIRG model will do better than Chung-Lu. It's also interesting to see how large a dimension d is necessary for a good GIRG fit.

As a first sanity check, we compare the MCMC fit $\hat{\theta}_{\text{GIRG}}$ to the more simply fit copy-weight Chung-Lu parameters $\hat{\theta}_{CL}$, by comparing $p(G|\hat{\theta}_{\text{GIRG}}, \mathcal{G}_{\text{GIRG}})$ vs $p(G|\hat{\theta}_{CL}, \mathcal{G}_{CL})$.

For most of the facebook graphs however, $p(G|\hat{\theta}_{\text{GIRG}}, \mathcal{G}_{\text{GIRG}}) < p(G|\hat{\theta}_{CL}, \mathcal{G}_{CL})$, despite GIRGs having more parameters / flexibility to fit G!

The GIRG model is far too confident about edges, giving $p_{uv} \approxeq 1$ for small $\|x_u - x_v\|$ and $p_{uv} \approxeq 0$ for large $\|x_u - x_v\|$. Hence a mistake on a few edges can lead to a large penalisation in likelihood. The Chung-Lu model is much more forgiving, with all probabilities more medium sized.

One reasonable tweak is to introduce a *failure rate* $0 \leq f \leq 1$ to the GIRG model:

$$p_{uv} = (1-f)\min\left\{1, c\left(\frac{w_u w_v / W}{\|x_u - x_v\|^d}\right)^\alpha\right\} \tag{5.6}$$

For a social network this means that two highly similar people are not guaranteed/forced to be friends. Indeed there may be a very like-minded person who lives next door to you that you've never met, or had no opportunity / free time to properly get to know.

A failure rate will lower the impact of short non-edges (mistakenly predicted with a high edge probability), but for long edges (mistakenly predicted with a low edge probablity) we need a baseline edge probability to prevent p_{uv} being too small. A simple fix is a *mixture probability* of deriving edges instead from the Chung-Lu model - i.e. letting

$$p_{uv} = \eta\ p_{uv}^{CL} + (1-\eta)\ p_{uv}^{GIRG} \tag{5.7}$$

for mixture probability $0 \leq \eta \leq 1$. This should not be seen strictly as an ensemble of models or gross addition to the number of parameters of the GIRG model, as the GIRG model already contains fit node weights $(\hat{w}_u)_{u \in V}$. The mixture probability η does also help a lot on short non-edges similarly to failure rate f, but it could still be good to have both as even the Chung-Lu model can demand an edge $u \sim v$ with high probability if w_u, w_v are both very large - though this is quite rare.

The intuitive social network interpretation of a GIRG Chung-Lu mixture model is that it allows for some "random" friendships - ones that wouldn't normally have

been predicted by the GIRG model. For instance if you happen to sit next to a stranger on a plane, as long as they don't fall into a sworn enemy category you may still add them as a facebook friend with a vague promise of staying in touch. Of course you and the stranger must bear at least some small similarity to have been on the same plane, but this would likely need a very high dimensional GIRG model to be fully captured.

With these two new parameters f and η, the augmented GIRG model does have a higher specific fit likelihood than the Chung-Lu model:

$$p(G|\hat{\theta}_{GIRG}, \mathscr{G}_{GIRG}) > p(G|\hat{\theta}_{CL}, \mathscr{G}_{CL}) \tag{5.8}$$

In a holistic MCMC setup these two parameters could also have a prior and be sampled from the posterior, but for simplicity we set them to $f = 0.3, \eta = 0.5$, which perform reasonably. See fig. 5.1 for some likelihood convergence curves on some example graphs. If we don't use a failure rate and mixture probability, the MCMC actually converges to a perverse set of node locations that are almost uniform, i.e. striving to be far apart, due to the high penalty cost of short non-edges.

5.2.2 Percent Edges Captured Metric Comparison

A good and simple framework to compare quality of fit without taking into account increased parametrisation is to analyse the "accuracy" on successfully producing edges / non-edges.

Our MCMC posterior is constrained by directly fitting $\hat{c}$ to produce a similar number of edges $|E|$ as in the real graph. Hence in the standard classification confusion matrix of true/false positives/negative edge classifications

TP	FN
FP	TN

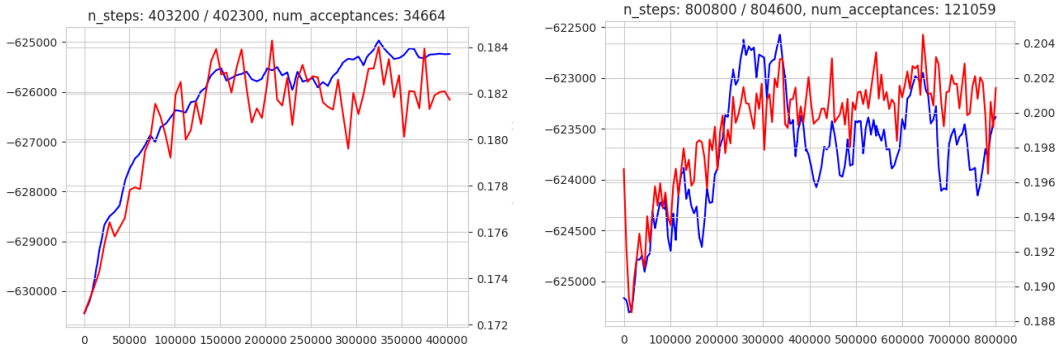
(5.9)

we can equivalently count $\frac{TP}{TP+FN}$ (Recall), the fraction of edges in the real graph that are successfully predicted, or $\frac{TP}{TP+FP}$ (Precision), the fraction of edges in the predicted graph that are also in the real graph, these numbers will be very similar. We call this metric *Percent Edges Captured (PEC)*, and focus on this instead of the alternative Percent Non-Edges Captured (PNEC). PEC is preferrable as our graphs are all relatively sparse - hence the PNEC is always high as it is dominated by the large number of non-edges.

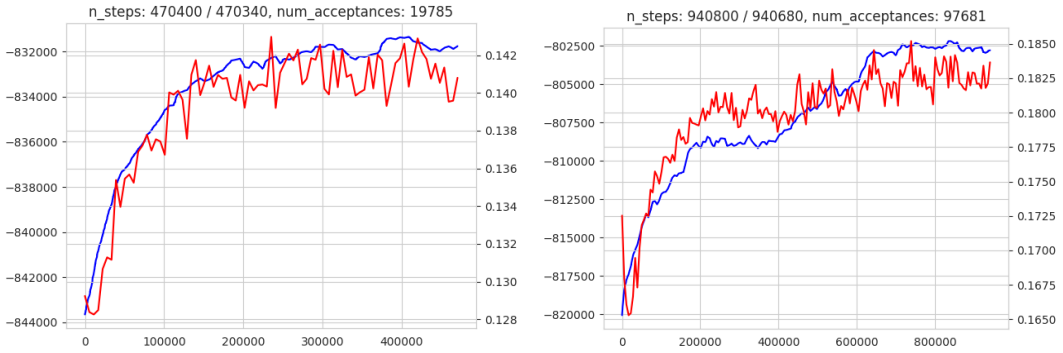
5.3 Direct Ordered Likelihood Maximisation

MCMC proved too slow and didn't achieve such high PECs. Hence we shifted to a slightly faster and simpler approach inspired by [García-Pérez et al., 2019]'s *Mercator* monikered method.

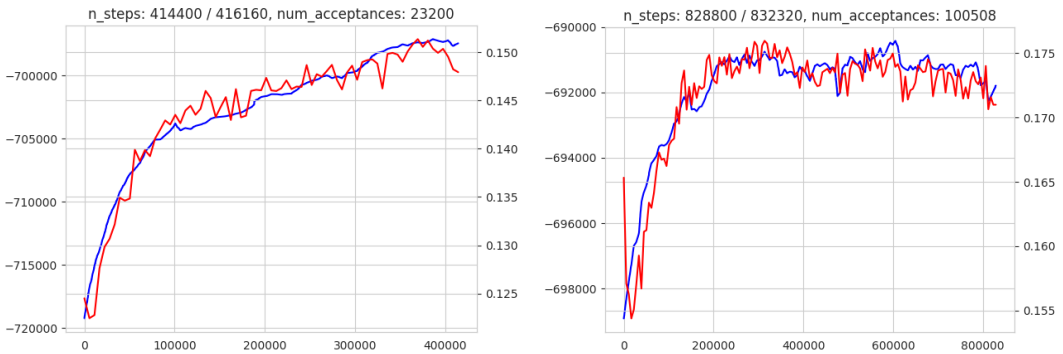
Instead of picking points to update randomly, we sequentially update the positions of every single node, ordered from highest to lowest degree; Mercator uses a more sophisticated ordering based on the *onion-decomposition* of the input graph. Updating point $\boldsymbol{x}_u$ is done by proposing 100 new locations (Mercator actually suggests $O(\log n)$) near to current location, near to a neighbour, or random uniform



(a) Amherst41 $d = 1$ **(b)** Amherst41 $d = 2$



(c) Trinity100 $d = 1$ **(d)** Trinity100 $d = 2$



(e) Hamilton46 $d = 1$ **(f)** Hamilton46 $d = 2$

Figure 5.1: MCMC runs for socfb-Amherst41, sofcb-Trinity100, socfb-Hamilton46 with failure rate 0.3 and Chung-Lu Mixin rate 0.5, with a cube GIRG model. Log likelihood wise, Amherst 1D GIRG does best; for Bowdoin it's 2D GIRG. x-axis is number of steps; left y-axis (blue curve) is log likelihood; right y-axis (red curve) is PEC metric

in the cube. Instead of using an acceptance probability as in MCMC, we simply pick the best of all the proposals, by marginal edge likelihood $\prod_{v \neq u} p(e(u,v)|...)$.

Updating points in order of highest degree makes sense as in essence a high degree node "drags" its local lower degree neighbourhood of nodes along with it, and thus should be updated in that order - otherwise if its lower degree neighbours are moved first, they might spread out more nonsensically and leave the higher degree node no good place to move to. We think this happened to some extent with the MCMC method, evidenced by initial drops in the PEC in fig. 5.1 for 2d GIRG fits.

After each round of point updates, α is fit to maximise the likelihood, and c is refit to match the number of edges in the real graph, similarly as in section 3.2.4. Multiple rounds are repeated until convergence. One optimisation that Mercator implements but we missed was to also after each round update node weight estimates: comparing two nodes with similar degree, if the first one has more geometrically close nodes than the second, the second should have a relatively higher estimated weight.

5.3.1 Ideas for speeding up convergence

In fact the Mercator method doesn't do rounds of point updates, they are confident in node locations after just one round. [Boguná et al., 2010] similarly seeks to minimise computation by only doing iterative Metropolis-Hastings updates on higher degree nodes - fitting locations for lower degree nodes just once. Both papers deal with fittig HRGs; unfortunately higher dimensioned GIRGs have even more issues with speed of fitting locations due to an expanded search space. We present some ideas for speeding up convergence in a related vein that could be explored more in future work.

Gradient based point proposals When randomly proposing 100 (or $O(\log n)$) new locations for each node, the resultant likelihood contribution of each location has to be computed. The number of locations would likely need to be exponentially increased for larger dimensions d to achieve convergence. An cheaper alternative is to just have one good proposal which is not random, rather taken as a small step in the direction of the gradient of the node likelihood contribution with respect to its location. We did limited tests which indeed showed faster convergence, but in general to a lower PEC value than before. One issue seemed that a much lower learning rate (step length in gradient direction) might be needed for $d = 1$ compared to higher dimensions: for $d = 1$, a node has neighbours and non-neighbours to the left and right, and will end up with quite a strong gradient pointing in one of the two directions, whereas for $d \geq 2$ forces on the node are in all different directions and hence more likely to be more finely balanced out with a smaller resultant gradient. Gradient clipping could potentially help. Another issues is that without random new point location proposals, nodes could easily get stuck in local optima.

Sequential dimension fitting For real graphs, there is generally a steeper decrease in marginal benefit from adding dimensions than from strictly GIRG generated graphs. Hence a point fitting approach that is sequential in dimensions could make sense. We could first fit the first (and hence picked up as most important

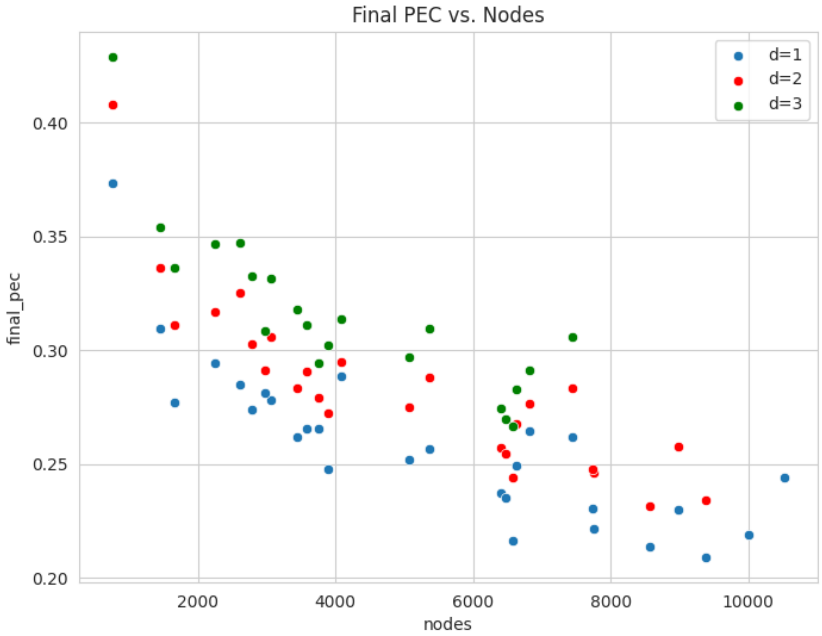


Figure 5.2: Final converged PECs for fitting 1d, 2d and 3d cube GIRGs to real-world Facebook graphs using direct ordered likelihood maximisation. Missing some 3d numbers as batch job exceeded allotted time.

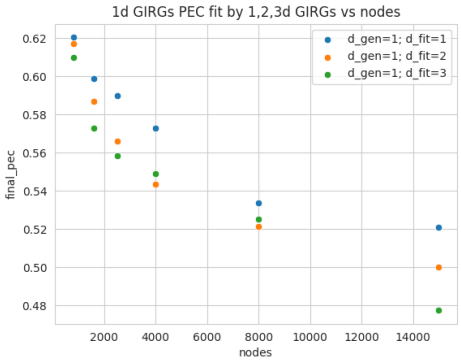
dimension) of every node. Then holding this fixed, fit the second dimension of every node, and so on. In essence it switches an exponential in d search space scaling to be just linear in d. This is similar to an iterative eigenvalue algorithm.

This sequential dimension fitting would of course have similar local optima issues. A simple example is a $d = 2$ MCD GIRG setting with three nodes A, B, C, where $A \sim B;\ B \sim C;\ A \nsim C$. Fitting the first dimension, the nodes would be placed in a line as A-B-C, to try and maximise the A-C distance while minimising A-B, B-C. This would unfortunately still lead A-C to be not so small due to the triangle inequality, which could not be improved when fitting the second dimension. The true optimal solution would be to have A-B close in the first dimension, B-C close in the second dimension, and A-C far in both dimensions.

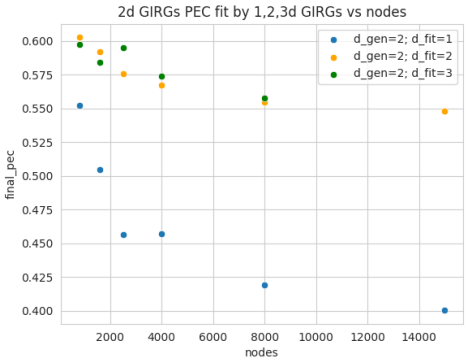
5.3.2 Analysis of quality of fit of GIRGs to real graphs

We see in fig. 5.2 the final achieved PECs of different dimensional cube GIRGs fit to real-world graphs. At first blush, PECs seem worryingly low at just 22% to 32%. You wouldn't be happy with a cat image classifier with only 30% accuracy! Furthermore, PEC decreases with increasing node count which seems to imply that our model fitting could be a fluke that only works on smaller graphs, and that might scale to irrelevance for even larger graphs. Luckily all these signs of disaster have good explanations, and we can mostly restore faith in our point fitting methods.

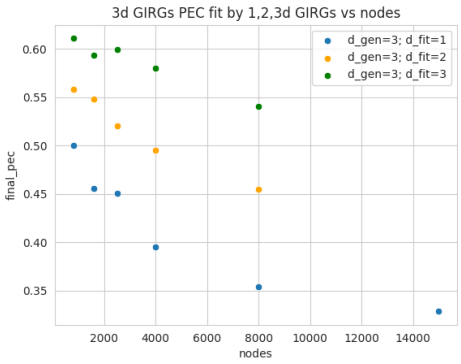
Inherently low GIRG PECs A first thing to note is that edge probabilities p_{uv} for a GIRG are generally not all $p_{uv} \in \{0, 1\}$ (at least for $\alpha < \infty$). Hence we cannot



(a) input graph is a 1d cube GIRG



(b) input graph is a 2d cube GIRG



(c) input graph is a 3d cube GIRG

Figure 5.3: Each plot shows the final PEC performance of fitting 1d, 2d and 3d cube GIRGs to an input graph of varying number of nodes. Input graphs are themselves generated from a cube GIRG - a different dimension $d = 1, 2, 3$ for each plot.

expect too high a PEC, even if we fit every single GIRG parameter perfectly to identically reproduce all the p_{uv}. We observe this in fig. 5.4, where PEC starts for the smallest graphs of $n = 800$ nodes at 64%. Note the plot is of cube GIRGs generated with $\alpha = 1.3$ which is quite typical an LCC fit on the real-world Facebook graphs, so indeed their edge probabilities range broadly in $[0.0, 1.0]$.

Fixed average degree $\implies$ lower GIRG p_{uv} $\implies$ lower PECs for higher n Also striking in fig. 5.4 is a similar curve to fig. 5.2 of decreasing PEC with increasing node count. We think this is an artifact of the fixed average degree of $\overline{deg} = 60$ used across these varying node count GIRGs. We fixed $\overline{deg}$ because the socfb graphs have basically stable average degrees - correlation coefficient of just 0.1 with node count, and their mean average degree across graphs is 77. We think that as n grows for a fixed $\overline{deg}$, since long distance edges in a GIRG contribute a constant fraction of total edges, in order to keep average degree constant the edge probability constant c is lowered. Hence for small n all edge probabilities p_{uv} are generally larger and less numerous ($O(n^2)$ potential edges); for large n there are many more potential edges, and generally smaller edge probabilities p_{uv}. This naturally brings about decreased PEC for larger n.

Fixed average degree $\implies$ lower PECs in general Another issue with fixed average degree and increasing n is that even in the dumbest case of fitting an Erdos-Renyi graph to an input graph, we would observe a $1/n$ shaped PEC curve

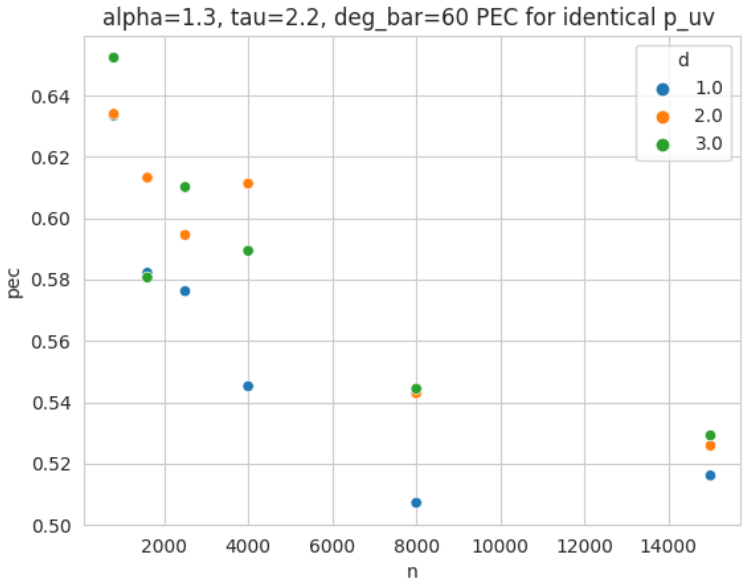


Figure 5.4: Exact p_{uv} replication has PEC $\in [0.52, 0.64]$, and decreasing with increasing node count. Done for $\alpha = 1.3, \tau = 2.2$ cube GIRGs of dimension $d = 1, 2, 3$ and average degree 60.

with increasing n. For each present edge in the input graph, there is a $p = \frac{\overline{deg}}{n}$ chance that it is also present in the Erdos-Renyi graph. Hence PEC would be $\frac{\overline{deg}}{n}$ in expectation.

This means that if we assume real-world graphs are only partially GIRG explainable, and that their residual non geometric random component is hence hard to fit with our low dimensional cube GIRGs, then the observed PEC against node count n plot would still look like a $1/n$ curve, but with a higher non-zero asymptote.

Chung-Lu control shows GIRGs not overfitting on real-world As a control, we fit GIRGs using direct ordered likelihood maximisation to Chung-Lu graphs, and observed PEC's is not much higher than between two resampled Chung-Lu graphs, at 10% for graphs of $n = 1000$ nodes, decaying to 3% for $n = 5000$. While there is roughly a 3% PEC boost from increasing dimension d which is clearly overfitting, this vindicates that the relatively higher acheived PECs of $22\% - 33\%$ in fig. 5.2 represent a true geometric quantity of real-graphs that we are successfully fitting.

Overall we find that cube GIRGs get quite a decent fit on the real-world facebook graphs, and that again 1 dimension seems already to bring a high amount of accuracy - increasing d only brings small gains, e.g. looking at graphs of size about $n = 8000$ nodes, the PEC increase from 1d to 2d cube GIRG fitting is just $22\% \to 25\%$. In contrast for fitting a 2d cube GIRG with a 1d cube GIRG, PEC jumps from $42\% \to 55\%$; fitting 3d with 1d PEC jumps from $0.35 \to 0.55$. We can observe in fig. 5.6 the final fit point locations of 2d GIRGs to a few graphs (2d because this is easiest for visualisation in a plot) - we see that points for socfb-Bowdoin47 and especially socfb-Haverford76 look like they could be pretty well approximated by 1d fit locations, whereas socfb-Rice31 has more complexity going on, and an extra dimension seems important to fully capture the graph structure.

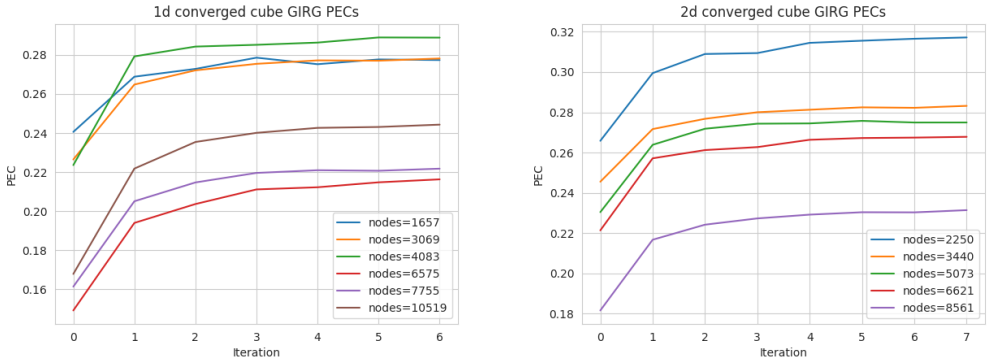
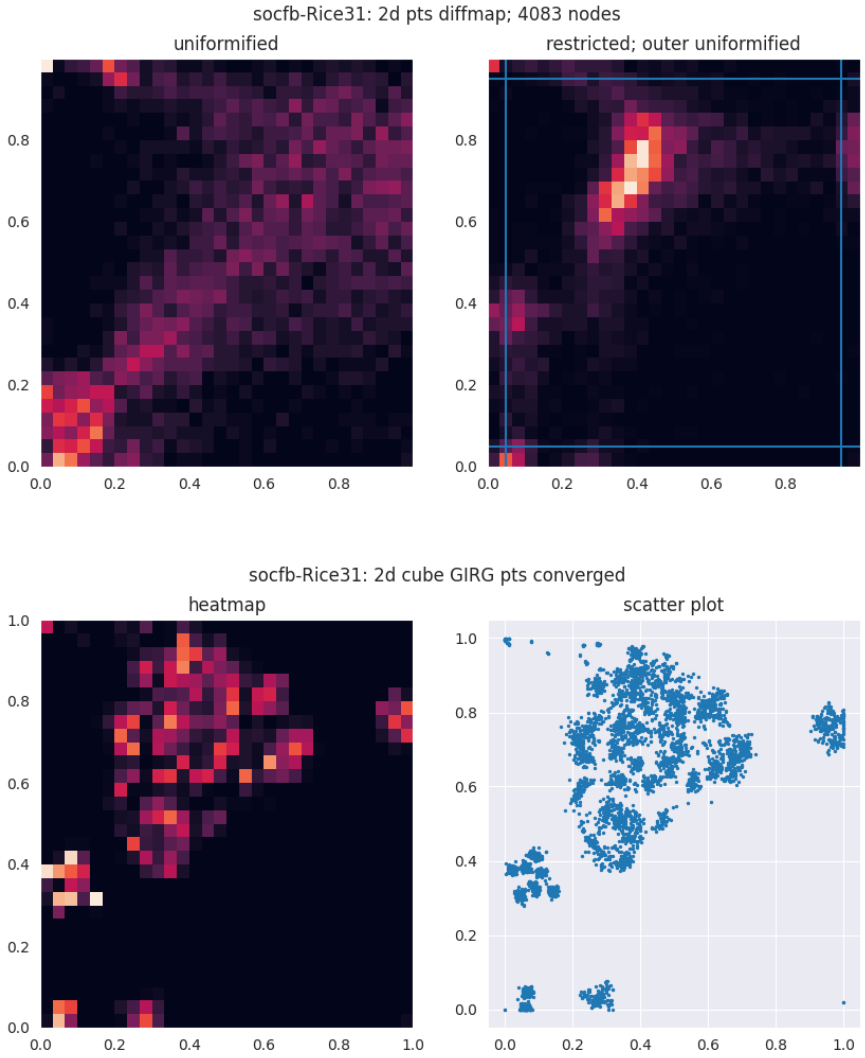
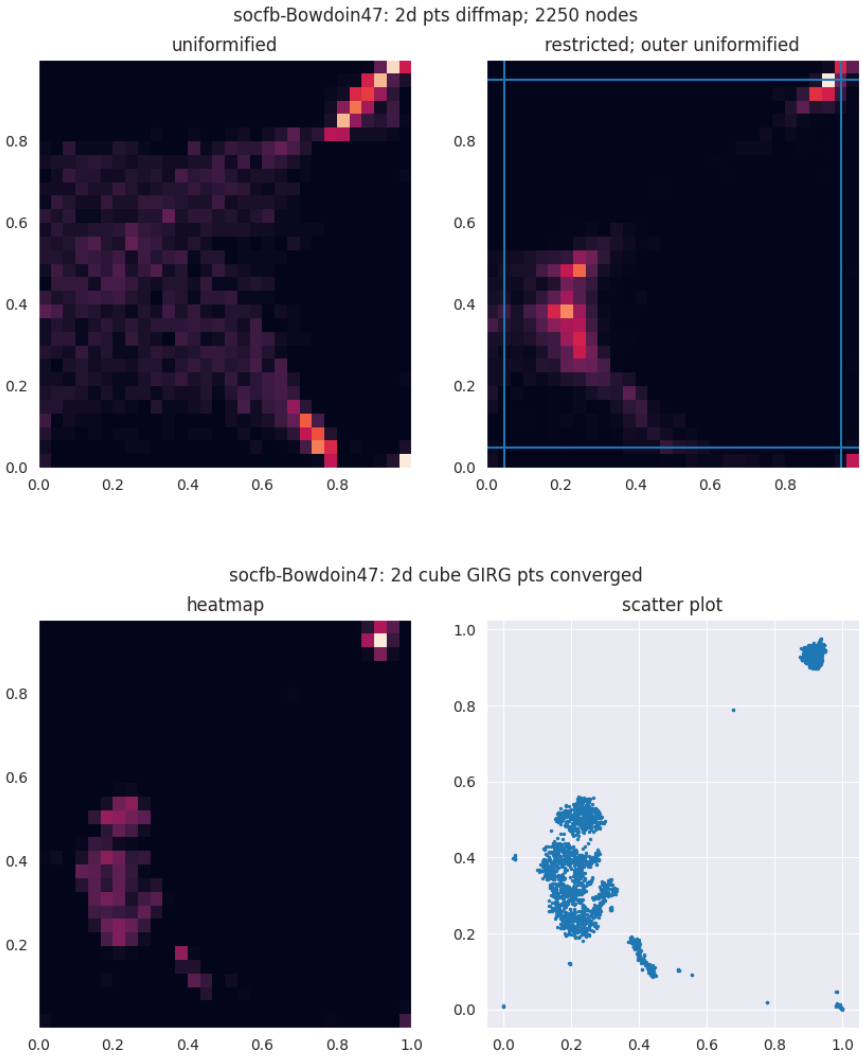
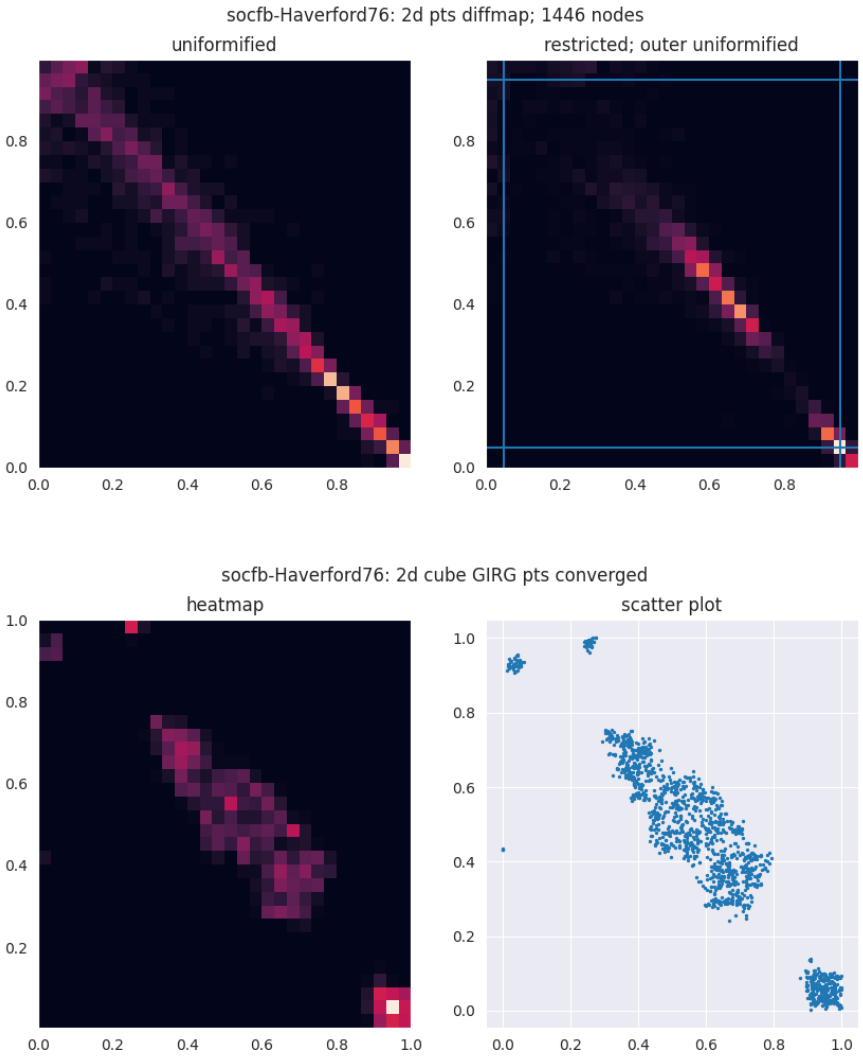


Figure 5.5: Direct ordered likelihood maximisation PEC convergence over iterations for a range of example socfb graphs with different number of nodes. Each curve is a single fit graph. X-axis is round number, where each round every node is moved once to a new location. Y-axis is PEC after that round. We claim convergence due to plateaued PEC, and note that in general the final achieved PEC decreases with graph node count.

Figure 5.6: 2d cube GIRG fitting examples, to three different Facebook graphs. Two initial 2d diffusion map embeddings are shown with either uniformify post-processing or restricted-rescaling. The blue lines for the restricted version show the border at which points are outer uniformified. Bottom two plots are heat map and scatter plots for the nodes having undergone further point location refinement using direct ordered likelihood maximisation. Note for example for socfb-Haverford76, restricted-rescaling separated the points into two (maybe even more) linear clusters which is a better initialisation, whereas uniformify was incapable of this and just produced one contiguous line of points. This also occurs to some degree for the other two graphs.




socfb-Bowdoin47: 2d pts diffmap; 2250 nodes
uniformified
restricted; outer uniformified
socfb-Bowdoin47: 2d cube GIRG pts converged
heatmap
scatter plot


socfb-Haverford76: 2d pts diffmap; 1446 nodes
uniformified
restricted; outer uniformified
0.0
0.2
0.4
0.6
0.8
socfb-Haverford76: 2d cube GIRG pts converged
heatmap
scatter plot
0.0
0.2
0.4
0.6
0.8
1.0

Chapter 6

Conclusion

The goal of this thesis was to explore the capacity of different GIRG model variants to match a dataset of real social network graphs, due to their suspected inherent geometrical generation and power law tailed degree distribution. We pursued this in two relatively distinct efforts:

1. using the high level framework of [Bläsius et al., 2018a] in chapter 3 to assess realism of different generative models on various global graph statistics
2. exploring methods to fit GIRG node weights and locations for an input real graph in chapter 4 and chapter 5

There was also a third effort, to use graph kernels as an alternative to Bläsius's framework for evaluating a generative model's realism, which was unsuccessful, and documented in appendix A.1.

1. showed GIRGs to perform similarly or better in realism than other simpler generative models, with different types of GIRGs excelling in simulating different statistics of the real graphs. 2. showed more concretely that GIRGs are expressive enough to replicate the connectivity of these social networks. Both 1. and 2. showed that $d > 1$ dimensional GIRGs generally did not improve significantly on 1d GIRGs, but suggested that higher dimensions might work best in a mixed min/max GIRG or distorted GIRG framework where there's a natural downwards progression of influence of dimensions.

Future Work The different GIRG variants that we have covered have only recently risen to prominence, and future work might compare them more thoroughly, both practically in relation to real graphs and theoretically. Other real graph datasets than the Facebook social networks we used might be explored, particularly ones where ground truth node attributes are known. These might provide a geometric influence on edges, and be compared to fit GIRG node locations. Adapting the GIRG uniform location priors and power law weight priors to something more flexible and realistic to real-world data could also be fruitful.

Many thanks to my supervisors Marc Kaufmann and Ulysse Schaller, as well as Johannes Lengler and Afonso Bandeira, for their valuable input and guidance throughout the thesis. Thank you to Angelika Steger and her lab for their friendly and welcoming research environment. Finally much appreciation to friends, family,

ETH, etc. for their support and existence. And thank you to the reader for making it this far! I hope it has been an enjoyable / interesting read.

Appendix A

Dummy Appendix

Contents

You can defer lengthy calculations that would otherwise only interrupt the flow of your thesis to an appendix.

A.1 Graph Kernels

In this section we try to compare the bayesian evidence based plausibility of different GGMs in generating Facebook graphs by using the method of *graph kernels*. Unfortunately the whole endeavour was unsuccessful, but we nonetheless briefly detail the framework and our experiments.

A.1.1 Bayes Factor Theory

A bayesian approach to model selection is to compare the posterior probability of different models $p(M_1|D)$ vs $p(M_2|D)$, given the model generative processes $P(D|M)$. M_1, M_2 are possible models of the data, e.g. $M_1 = \mathscr{G}_{1D\text{ GIRG}}$ and $M_2 = \mathscr{G}_{CL}$. D is a single graph instance G, or alternatively a whole dataset of our 100 Facebook graphs, assuming that they all come from the same generative model family. We have some prior of $p(M_1)$ vs $p(M_2)$, e.g. $50:50$. We could even possibly have a set of models $M_{d=1}, M_{d=2}, M_{d=3}, ...$ for different dimensional GIRGs with some kind of decaying $p(1d\text{ GIRG}) > p(2d\text{ GIRG}) > p(3d\text{ GIRG}) > \dots$.

For now focussing on just M_1 vs M_2,

$$p(M_1|D) = \frac{p(D|M_1)p(M_1)}{p(D)} \implies \frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)p(M_1)}{p(D|M_2)p(M_2)} \tag{A.1}$$

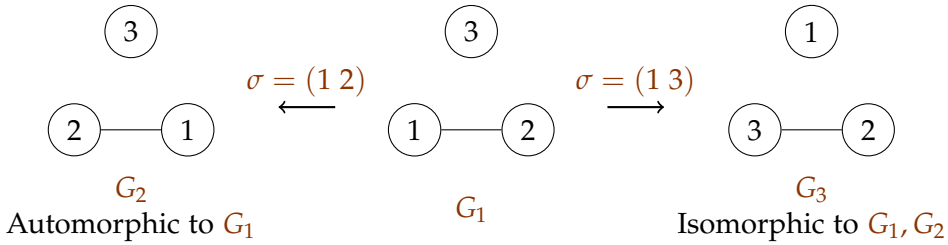


Figure A.1: Example automorphic/isomorphic graphs

Model selection is done with the ratio $\frac{p(M_1|D)}{p(M_2|D)}$ which is called the *Bayes Factor*. In order to compute the bayes factor, we will for now ignore the priors $p(M_1), p(M_2)$, and hence focus just on the likelihoods $p(D|M_1), p(D|M_2)$.

A.1.1.1 Graph isomorphism problem in computing model evidence

In our case if M_1 is a 1D GIRG, and we've decided to fix $\alpha, \{w_u\}_{u\in V}$ and just vary $\theta = c, \{\boldsymbol{x}_u\}_{u\in V}$, then we could Monte Carlo sample $\theta \sim p(\theta)$ from the prior to get an estimate for

$$p(D|M_1) = p(G|\mathscr{G}_{\text{1d-GIRG}}) = \int_\theta p(G|\theta, \mathscr{G}_{\text{d-GIRG}}) p(\theta|\mathscr{G}_{\text{d-GIRG}}) d\theta \tag{A.2}$$

The simplified notation is dropping the $|M_1$ as we fix into some dimensional GIRG universe, $\int_\theta p(G|\theta)p(\theta)d\theta$.

There's a problem here. Since our data is a graph $D = G$, in the computation we actually need to inspect $p(G|\theta) = \sum_\sigma p(G' \overset{\sigma}{\cong} G|\theta)p(\sigma)$. Here σ is a permutation of the node ID, and $\overset{\sigma}{\cong}$ denotes *graph isomorphism* under this permutation. Note here our abuse of notation - normally $p(G|\theta)$ is not a sum over permutations, but rather the probability of generating this exact graph G; this will have to be contextually apparent. So instead of evaluating $p(G|\theta)$, we rather want to evalute "the probability of producing G', given parameters θ, which is isomorphic to G".

Small concrete example of a labelled graph $G_1 = (V, E) = (\{1,2,3\}, \{(1,2)\})$ which is shown in fig. A.1. G_1 is isomorphic to any other labelled graph H if they look the same if you anonymise the nodes. More formally, if $\sigma : V(G) \to V(H)$ is a bijection mapping nodes to nodes, such that $u \sim v$ in $G \iff \sigma(u) \sim \sigma(v)$ in H, then G is isomorphic to H with permutation σ, written $G \overset{\sigma}{\cong} H$. So G_1 is isomorphic to G_3 with edge set $\{(3,2)\}$. You can even say that it is automorphic to the graph H with edge set $\{(2,1)\}$, i.e. just switching nodes $1, 2$. On this graph G_1 of 3 nodes, there are $3! = 6$ node ID permutations, each of which produces an isomorphic graph G'. For our particular G_1, we could group together these 6 isomorphic graphs G' into 3 pairs of automorphic graphs.

Finally repeat for further θ samples and take an outer average.

Consider a simplified Chung-Lu / GIRG model of three nodes where of $\theta = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ and all nodes have the same weight 1. The Chung-Lu has identical $p(G')$ for each of the 6 ismorphisms. The GIRG model does not - if $\boldsymbol{x}_1, \boldsymbol{x}_2$ are close, and

x_3 is far from both, then it awards higher probability to $G' = (V, \{(1,2)\})$ and $G' = (V, \{(2,1)\})$. Furthermore both models always award the same probability to any equivalent class of automorphic graphs - this is because the adjacency matrix is the same for automorphic graphs, which is all that the models care about.

Unfortunately computing the $n!$ length sum $\sum_\sigma p(G' \overset{\sigma}{\cong} G|\theta)p(\sigma)$ is infeasible for all but tiny graphs. As an alternative, we hoped to use a graph similarity kernel k which compares two graphs G, H, giving some measure $k(G,H)$ of how similar they are up to ismorphism (in a node permutation invariant way). Therefore the idea would be to replace the

$$\text{replace the incorrect} \quad p(G|\mathscr{G}_{\text{GIRG}}) = \int_\theta p(G|\theta)p(\theta)d\theta \tag{A.3}$$

$$\text{with} \quad E_{G' \sim \mathscr{G}_{\text{GIRG}}}[k(G,G')] \approxeq \frac{1}{m}\sum_{i=1}^{m} k(G, G'_i \sim \mathscr{G}_{\text{GIRG}}) \tag{A.4}$$

which is Monte Carlo computable.

A.1.2 Graph Kernel Introduction

Graph Kernels provide a means for a similarity metric between graphs. They're ideally a positive semidefinite function $k : \Gamma \times \Gamma \to \mathbb{R}$, where Γ is the set of all graphs. Such a function exists if and only if there is a corresponding feature map resentation of $\phi : \Gamma \to \mathcal{H}_\phi$ where $\mathcal{H}_\phi$ is a Hilbert space, and $k(G,G') = \langle \phi(G), \phi(G') \rangle_{\mathcal{H}_\phi}$ is just an inner product.

Graph kernels can be contrasted with the Blasius' GGM realism framework . Blasius compares multiple different graph feature combinations on which to train an SVM for distinguishing two graph datasets. Viewing the graph kernel as an inner product of graph feature maps, these hopefully encapsulate all the relevant information about the graph including the specifically chosen features in Blasius' framework like graph diameter, average degree, etc. The question of which chosen graph features to consider then shifts to which graph kernel to use!

There are a range of graph kernels to choose from, however given the relatively large size of our graphs, runtime can be an issue. We ended up testing two kernels; alas both proved unsatisfactory. Without further expertise on graph kernels, we were forced to abandon this line of attack. We think that graph kernels might be more effective on smaller graphs with more unique structure (our random graph models and the real graphs themselves have a lot of homogenous structure) and particularly graphs with meangingful node labels.

A.1.2.1 Random walk kernel

The *random walk kernel* between two graphs is just a count of the number of common walks in the pair, of any length l. Thißs is generally an infinite sum, so geometric weighting with the factor λ^l is used to decay the contribution of longer walks, where $0 < \lambda < 1$ (we tested with $\lambda = 10^{-5}$).

The product graph of G and G' is defined as

$$G_\times = (V_\times, E_\times) \quad \text{where} \quad V_\times = \{(v,v')|v \in V, v' \in V'\} \tag{A.5}$$

$$\text{and} \quad E_\times = \{((v_1,v'_1),(v_2,v'_2))|v_1 \sim v_2 \in E_1, v'_1 \sim v'_2 \in E_2\} \tag{A.6}$$

This definition can be extended to node-labelled graphs, where we only take product vertices $(v, v') \in V_\times$ if $v \in V, v' \in V'$ have the same label. Common walks in the pair of graphs G, G' are just walks in the product graph.

The naive implementation of the random walk kernel has complexity $O(n^6)$, however this can be sped up to $O(n^3)$. This is still too slow for our purposes. We only made limited tests with small graphs of $n \leq 3000$ nodes - see e.g. fig. A.2b.

A.1.2.2 Weisfeiler-Lehman kernel

The *Weisfeiler-Lehman kernel* runs much faster in $O(h|E|)$ time, where $|E|$ is the number of edges in the graph and h is a chosen parameter, the number of rounds of relabelling within the algorithm. The algorithm requires some starting node labels $(l(v))_{v \in V}$ - we colour nodes into a small discreet set of colours by grouping together nodes with similar sized degrees. It also uses a base kernel k_{base} which is computed at each relabelling round - we used the simplest/speediest node label *histogram dot product kernel*:

$$k(G, G') = \langle \boldsymbol{f}, \boldsymbol{f}' \rangle \quad \text{where} \quad \boldsymbol{f} = (f_1, \ldots, f_k),\ f_i = |\{v \in V \ :\ l(v) = i\}| \tag{A.7}$$

The Weisfeiler-Lehman kernel is then defined as

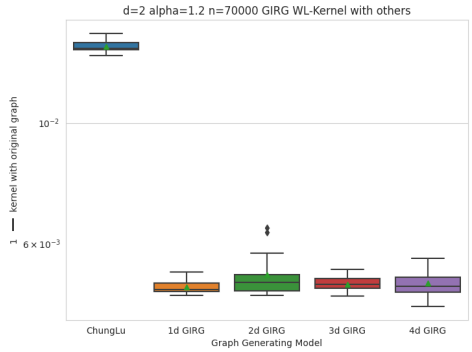
$$k_{WLK}(G, G') = k_{\text{base}}(G_1, G_1') + \cdots + k_{\text{base}}(G_h, G_h') \tag{A.8}$$

$$G = G_1,\ G' = G_1';\quad G_{i+1} \text{ by relabelling } G_i :\quad l(v) \leftarrow (l(v), (l(u))_{u \sim v}) \tag{A.9}$$

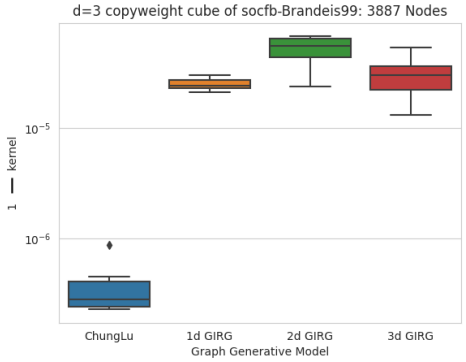
This looks odd, but in practice each label is just rehashed as a new integer rather than becoming a highly nested tuple.

A.1.3 Experiments

Unfortunately both kernels failed to pass the basic test of showing higher similarity between two graphs generated from the same GGM than from a different GGM. We tried a variety of combinations, but didn't get very reliable results: see fig. A.2a and fig. A.2b.



(a) Weisfeiler-Lehman Kernel of a d=2, alpha=1.2, n=70000 Torus GIRG with other generated graphs (13 per model). All the GIRGs are more similar to the original than Chung-Lu, which is good, but we cannot differentiate between GIRGs of different dimensions.



(b) Random Walk Kernel of a d=3 copy weight cube GIRG fit to socfb-Brandeis99 (matching number of edges and mean LCC), with other graphs generated from different GGMs (6 per model type). We hoped for 3d GIRGs to have the highest similarity to the input graph, however instead Chung-Lu graphs matched closest.

Bibliography

[Belkin and Niyogi, 2001] Belkin, M. and Niyogi, P. (2001). Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, 14.

[Berry and Harlim, 2018] Berry, T. and Harlim, J. (2018). Iterated diffusion maps for feature identification. *Applied and Computational Harmonic Analysis*, 45(1):84–119.

[Bläsius et al., 2021] Bläsius, T., Friedrich, T., and Katzmann, M. (2021). Force-directed embedding of scale-free networks in the hyperbolic plane. In *19th International Symposium on Experimental Algorithms (SEA 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[Bläsius et al., 2018a] Bläsius, T., Friedrich, T., Katzmann, M., Krohmer, A., and Striebel, J. (2018a). Towards a systematic evaluation of generative network models. In *Algorithms and Models for the Web Graph: 15th International Workshop, WAW 2018, Moscow, Russia, May 17-18, 2018, Proceedings 15*, pages 99–114. Springer.

[Bläsius et al., 2022] Bläsius, T., Friedrich, T., Katzmann, M., Meyer, U., Penschuck, M., and Weyand, C. (2022). Efficiently generating geometric inhomogeneous and hyperbolic random graphs. *Network Science*, 10(4):361–380.

[Bläsius et al., 2018b] Bläsius, T., Friedrich, T., Krohmer, A., and Laue, S. (2018b). Efficient embedding of scale-free graphs in the hyperbolic plane. *IEEE/ACM transactions on Networking*, 26(2):920–933.

[Boguná et al., 2010] Boguná, M., Papadopoulos, F., and Krioukov, D. (2010). Sustaining the internet with hyperbolic mapping. *Nature communications*, 1(1):62.

[Bringmann et al., 2016] Bringmann, K., Keusch, R., and Lengler, J. (2016). Average distance in a general class of scale-free networks with underlying geometry. *arXiv preprint arXiv:1602.05712*.

[Bringmann et al., 2019] Bringmann, K., Keusch, R., and Lengler, J. (2019). Geometric inhomogeneous random graphs. *Theoretical Computer Science*, 760:35–54.

[Coifman and Lafon, 2006] Coifman, R. R. and Lafon, S. (2006). Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30.

[Friedrich et al., 2023] Friedrich, T., Göbel, A., Katzmann, M., and Schiller, L. (2023). Cliques in high-dimensional geometric inhomogeneous random graphs. *arXiv preprint arXiv:2302.04113*.

[García-Pérez et al., 2019] García-Pérez, G., Allard, A., Serrano, M. Á., and Boguñá, M. (2019). Mercator: uncovering faithful hyperbolic embeddings of complex networks. *New Journal of Physics*, 21(12):123033.

[Krioukov et al., 2010] Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., and Boguná, M. (2010). Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106.

[Rossi and Ahmed, 2015] Rossi, R. and Ahmed, N. (2015). The network data repository with interactive graph analytics and visualization. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29.

[Singer, 2006] Singer, A. (2006). From graph to manifold laplacian: The convergence rate. *Applied and Computational Harmonic Analysis*, 21(1):128–134.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.