Title of Thesis

Master Thesis

S. Tudent

January 19, 2038

Advisors: Prof. Dr. A. D. Visor, Dr. P. Ostdoc

Department of Computer Science, ETH Zürich

Abstract

This example thesis briefly shows the main features of our thesis style, and how to use it for your purposes.

Contents

Chapter 1

Introduction

1.1 Thesis Introduction

Networks / Graphs are a useful model to describe a group of interconnected entities, and is a common abstraction of real life systems. Real world graph data abound, but often bear similarities in their structure. Different graph generating models have been proposed with justifications of reasonable formative mechanisms and likeness to real graphs. Theoretical study of these models bring insights on the real networks, and have use in generating synthetic graphs for testing purposes.

Geometric Inhomogenous Random Graphs (GIRGs) are one such graph generating model, first proposed in [Bringmann et al., 2016] as a generalisation of *Hyperbolic Random Graphs*. They permit great flexibility on the geometric component of the generative structure and hence can better model a wider range of real graphs. Their definition embodies the intuitive sense present in many graphs that nodes are more likely to be connected if they are "close together". GIRGs also have an inhomogenity between nodes that results in a power law degree distribution, which is observed in many real graphs.

In this thesis we assess how well GIRGs can simulate a dataset of 104 social network graphs downloaded from the Network Repository [Rossi and Ahmed, 2015]. These graphs are all Facebook networks from various towns in the US, so are structurally similar, and range from $n = 1000$ to $n = 100,000$ nodes. In chapter 3 we judge the mostly unmodified GIRG prior distribution of graphs on their high level similarities to the real graphs, and in chapter 5 a more intensive posterior style of fit GIRG in replicating real graphs on a node/edge level.

The first task of GIRG prior realism assessment is done in In chapter 3 using a high level feature dataset classification comparison framework proposed in [Bläsius et al., 2018]. Our results provide evidence that GIRGs are a more realisitc candidate 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 different GIRG variants for realism on these features, finding that while max norm torus GIRGs are a good baseline, cube GIRGs outperform on graph effective diameter, and mixed / minimum component distance GIRGs do best on closeness centrality. We also in most (but not all) cases find that a small geometric dimensionality of $d = 1$ already provides near peak GIRG realism, with little benefit from GIRGs of higher dimension.

We additionally try in chapter 6 to apply the method of *graph kernels* as an alternative framework to do model selection between the GIRG prior model and other generative graph models with respect to our real graph dataset. Unfortunately we failed to produce reasonable results.

For the second task of GIRG posterior fitting we explore the method of diffusion maps in chapter 4, in addition to either a markov chain Monte Carlo or maximum likelihood method of fitting geometric locations of nodes in the graph in chapter 5. We detail an empirical method using diffusion maps to find a reasonable dimension of GIRG. The full fitting procedure shows decent performance in replicating real graphs, and again supports the hypothesis that 1 dimension is mostly sufficient for this dataset of social networks.

Supporting all objectives, we also cover the different types of GIRGs and their selling points, and show practical methods for sampling and working with them.

Chapter 2

GIRGs - variants and generation

Contents

2.1 GIRGs definition

We outline the key elements of GIRGs and the main variations and how they fit into a wider context of random graph models.

The GIRG definition according to [Bringmann et al., 2019] is a random graph model defined by the edge connection probabilities

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

where $(w_u)_{u \in V}$ are node specific weights and $(x_u \in \chi)_{u \in V}$ are positions in some geometric space χ, generally taken to be the d-dimensional torus [1] $\mathbb{T}^n$, or the d-dimensional unit cube $[0,1]^d$. The normalising factor of W^{-1} where $W = \sum_{u \in V} w_u$

---

[1] The d-dim torus also looks like $[0,1]^d$, however opposite faces of the cube are identified. Think Pacman. Or donuts. Just not Pacman eating a donut...

ensures that the expected degree of a node u is unchanged even as $n \to \infty$; though u will have more possible other nodes to connect to, this is counterbalanced by the diminshing probability of connecting to any one.

2.1.1 Geometry

The geometric space χ, and the distance function $\|\cdot\|$ (not necessarily a norm) can vary a great deal without affecting key properties of GIRGs.

The edge probability p_{uv} has a geometric factor from the distance $r_{uv} = r(x_u, x_v) = \|x_u - x_v\|$. This inversely scales the probabilities so that nearby points (small r_{uv}) are more likely to have an edge than those further apart. Depending on the space χ and the function r chosen, it may be a norm, a metric, or not even. r_{uv} is taken to the dth power so that the number of edges is consistent across different values of d. Such proximity influenced p_{uv} is one way of bringing about the phenomenon of *clustering*, common in many real life graphs, whereby subgroups of nodes might have more edges within themselves than expected by chance.

Taking the distance function as the max norm, i.e. $\|x\| = \|x\|_\infty = \max_i |x_i|$ is also nice for proofs, but can be replaced by euclidean or other norms - by the equivalence of norms there isn't much difference. Note on the torus the max norm isn't strictly a norm, rather we abuse notation to write:

$$\|x - y\|_\infty = \max_i |x_i - y_i|_C \tag{2.2}$$
$$|x_i - y_i|_C := \min(|x_i - y_i|, 1 - |x_i - y_i|) \tag{2.3}$$

The *minimum component distance* (MCD) defined as $\|x\|_{\mathrm{mcd}} = \min_i |x_i|$ doesn't even satisfy the triangle inequality, but still retains most of the GIRG properties. This can make sense if we want two nodes to have a higher edge probability if they are close in at least one dimension, rather than every dimension.

$\alpha \in (1, \infty]$ is another parameter affecting the geometry. The edge probability formular distinguishes *short edges* where $r_{uv} < (w_u w_v / W)^{1/d}$ and $p_{uv} = \Theta(1)$ (for these the $\min(\cdot, 1)$ takes affect to ensure $p_{uv} \leq 1$), and *long edges* where $r_{uv} > (w_u w_v / W)^{1/d}$, and p_{uv} decays to zero with increasing distance. Larger values of α speed this decay, so more strongly restrict the number of longer distance edges. $\alpha = \infty$ enforces a sharp cutoff to $p_{uv} = 0$ for long edges.

2.1.2 Similarity to Chung-Lu

A key property of the GIRG model is that it's a geometric variant of the Chung-Lu model. The Chung-Lu model decrees edge probabilities between nodes for a given node weight sequence $(w_u)_{u \in V}$ as $p_{uv} = \Theta(\min\{1, \frac{w_u w_v}{W}\})$.

In a GIRG, geometry influences edge proabilities, however when you consider any two nodes u, v, marginalising over all of their possible locations (where they're close by or far apart), you get $E_{x_v}[p_{uv} | x_u, w_u, w_v] = \Theta(\min\{1, \frac{w_u w_v}{W}\})$, precisely like Chung-Lu just forgetting node locations $(x_u)_{u \in V}$.

Therefore properties/results about Chung-Lu generated graphs carry through automatically to GIRGs, like simple facts that $E[d_u] = \Theta(w_u)$ (the expected degree of a node u with weight w_u is proportional to w_u).

This seems strange as we have the $(w_u w_v / W)^\alpha$ term in the edge probabilities, however as introduced above, α only affects the decay rate of edge probabilities

for long edges, which is only a constant fraction of total edge probability, as long as $\alpha > 1$. If α were smaller we would actually blow up the degree d_u far beyond w_u, as the large number of potential long edges would have too many realised as actual edges, and dominate the short edges.

2.1.3 Power Law Degree Sequence

The weight sequence of a GIRG is 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 a power law tail degree sequence, since node expected degrees in the GIRG are proportional to their weight.

$$
\begin{aligned}
&\text{w} \sim \text{powerlaw}(\tau) && \text{exact power law distribution} \\
&p(w) \propto w^{-\tau} \text{ for } w \in [x_{\min}, \infty] && \text{pdf (default } x_{\min} = 1) \\
&p(\text{w} \geq w) \propto w^{1-\tau}
\end{aligned}
$$

The power law distribution is heavy tailed (heavier than exponentially decaying tails). $\tau > 2$ is important to ensure that $E[\text{w}] = \Theta(1)$, which means that although we may have some very large w_i in a sequence $w_1, ..., w_n$, still the majority of the total weight is in smaller valued w_i's.

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:

$$
p_{uv}(r) = \min\left\{1, c\left(\frac{w_u w_v/W}{Vol(r_{uv})}\right)^\alpha\right\}; \quad \rho_{uv} = \frac{w_u w_v/W}{Vol(r_{uv})} \tag{2.4}
$$

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

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

$$
p_{uv} = \min\left\{1, c\left(\frac{w_u w_v/W}{\|x_u - x_v\|_\infty^d}\right)^\alpha\right\} \tag{2.5}
$$

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 GIRG definition as p_{uv} always lies in the interval

$$c \min \left\{ 1, \left(\frac{w_u w_v / W}{\|x_u - x_v\|_\infty^d} \right)^\alpha \right\} \le p_{uv} \le \min \left\{ 1, \left(\frac{w_u w_v / W}{\|x_u - x_v\|_\infty^d} \right)^\alpha \right\} \quad (2.6)$$

if $c \le 1$, 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: such a GIRG has expected number of edges $\Theta(n)$, 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 GIRG Generation 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 $d \ge 4$. 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. The basic steps are:

1. sample $O(n)$ node weights w_u from a power law distribution

2. sample $O(n)$ node locations 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 Toroidal 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 of the torus, for the ability to hopefully generate more realistic graphs, as few situations in the real world are Toroidal.

For example with our socfb social network graphs, likely geometric features for each individual are home address, age, athletic proclivity, political leaning etc. In the environ of a town/city, home address is a 2D vector [2]. Age and athleticisim 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

---

[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

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 of normie to hipster.

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,

$$\text{in the torus} \quad Vol(r_{uv}) = (2\,\|x_u - x_v\|_\infty)^d \tag{2.7}$$

$$\text{now in the cube} \quad Vol(B_{r_{uv}}(u)) = Vol(B_{r_{uv}}(u)) \cap [0,1]^d \tag{2.8}$$

i.e. volume only counts within the cube itself, and hence oftentimes $Vol(B_{r_{uv}}(u)) \neq Vol(B_{r_{uv}}(v))$.

We will use a simpler cube GIRG formulation based solely on distance, i.e. the factor of r_{uv}^d in p_{uv}. This is conceptually easy to understand, and allows for a neat coupling with the original Torus geometry whereby $r_{uv}^{\text{cube}} \geq r_{uv}^{\mathbb{T}}$, with most pairs having $r_{uv}^{\text{cube}} = r_{uv}^{\mathbb{T}}$. We can say that a GIRG in cube geometry is stochastically dominated by its torus geometry counterpart as $p_{uv}^{\text{cube}} \leq p_{uv}^{\mathbb{T}}$, and is hence strictly sparser.

A volume based formulation of edge probabilities is more complicated - it 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 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 generation algorithm that couples to a generated torus GIRG. All that is needed are the original weights and point locations, and the edges present in the torus GIRG. The pseudocode is shown in algorithm 1. This runs in $O(n)$ extra time on top of the initial torus GIRG generation.

The coupling works for any pair of torus/cube distance functions $r_{uv}^{\text{cube}} \geq r_{uv}^T$, which results in stochastically dominated edge probabilities $p_{uv}^{\text{cube}} \leq p_{uv}^{\mathbb{T}}$. Thus 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. We just have to inspect each of the $\Theta(n)$ edges in the 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 via coupling

Require: n, d, c, τ, α
  $(G, \{x_u\}_{u \in V}, \{w_u\}_{u \in V}) \leftarrow \text{torus-GIRG}(n, d, c, \tau, \alpha)$
  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 max norm $\|\cdot\|_\infty$, euclidean norm, and the minimum component distance (MCD) $\|\cdot\|_{\text{mcd}}$ as alternative distance functions. While the max norm is commonly used in e.g. [Bringmann et al., 2019] as its 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 like finding a romantic 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 in the local football club, and sing in the 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.

$$\|x - y\|_{\text{mix}} = \min\left(\|x_1 - y_1\|_\infty, \|x_{[2,3]} - y_{[2,3]}\|_\infty, \|x_{[4,5]} - y_{[4,5]}\|_\infty\right) \quad (2.9)$$

This parses as points x, 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 min/max mixed GIRG.

2.4.1 Volume Formula for MCD, and Mixed Norms

The formula for $Vol(r)$ differs by the distance function used $r(x, y)$. For the max norm in d dimensions we saw that $Vol(r) = (2r)^d$ as 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 a ball of radius r - e.g. $2r; \pi r^2; , \frac{4}{3}\pi r^3$ for $d = 1, 2, 3$ respectively.

For the MCD, we can derive a formula using independent events. Consider the ball $B_r(x)$ and $y \sim U(\mathbb{T}^d)$ sampled uniformly from the torus. Let event A_i be

that $|y_i - x_i| < r$. Then $Vol(r) = P(\bigcup_{i=1}^d A_i)$. Since events A_i are independent by orthogonality of the individual dimensions, we have that

$$Vol(r) = P\left(\bigcup_{i=1}^d A_i\right) = 1 - P\left(\bigcap_{i=1}^d \overline{A_i}\right)$$
$$= 1 - \prod_{i=1}^d P(\overline{A_i}) = 1 - (1 - 2r)^d$$

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

$$r(\boldsymbol{x}, \boldsymbol{y}) = \min\left(\|\boldsymbol{x}_{I_1} - \boldsymbol{y}_{I_1}\|_\infty, ..., \|\boldsymbol{x}_{I_k} - \boldsymbol{y}_{I_k}\|_\infty\right)$$
$$\text{then } Vol(r) = 1 - \left[\prod_{i=1}^k 1 - (2r)^{|I_i|}\right]$$

2.4.2 MCD GIRG speedy combination of 1d GIRGs generating algorithm

We saw in section 2.2 that the max torus GIRG implementation of [Bläsius et al., 2022] has a runtime of $O(n)$ for fixed d, but this scales exponentially in d. We propose a faster algorithm that basically generates an MCD GIRG by generating d 1d GIRGs each in $O(n)$ time, and combining them together into one graph in total $O(dn)$ time. This gives a significant advantage to using MCD GIRGs over max GIRGs already for dimensions as low as $d = 4$.

As an example 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 independtly 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 we don't have to significantly over generate many edges in the 1d GIRGs to produce our MCD GIRG.

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 $\alpha = \infty \implies p_i \in \{0, 1\}$, so in particular $u \overset{3}{\sim} v \implies u \overset{2}{\sim} v \implies u \overset{1}{\sim} v$. 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 we rather must additionally check that any final edge does come from the smallest r_i graph. 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 $u \nsim v$ 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 edge probabilities corresponding to those in the resultant MCD GIRG. This means taking

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

rather than the standard 1d GIRG edge probability which just has $2r_i$ in the denominator. This pleasingly gives us that $p_i \leq p_{uv}$, since $r_{uv} := \min_j r_j \leq r_i$ where p_{uv} and r_{uv} are the edge probability, node pair distance respectively in the final 3d MCD GIRG. This means that in expectation the number of edges in each 1d GIRG is fewer than in the resultant MCD GIRG, i.e. $O(n)$ - this matches our "union of 1d GIRGs" intuition.

Extension to max GIRGs - We would like to use this higher dimension GIRG generating algorithm for max GIRGs as well. 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 $\Theta(n)$ edges. In short, now $r_{uv} := \max_j r_j \geq r_i$, so each 1d GIRG in expectation has more edges than the resultant max GIRG.

TODO the Blasius algorithm is "O(n)". It's definitely at least $O(|E|)$, but is it worse?

To generate the individual 1d GIRGs, we need at least complexity in their number of edges $O(|E|)$ which is no longer $\Theta(n)$. We give a sketchy intuition that can be made more rigorous:

Focusing on the $\Theta(1)$ fraction of edges in the resultant max GIRG which are both short: $p_{uv} = 1$ due to $c\left(\frac{w_u w_v / W}{(2r)^d}\right) \geq 1$, and between nodes of small weight e.g. $w_u, w_v \in [1, 2]$. Then for a fixed average degree $\overline{\deg}$ of the max GIRG, we can roughly say that the neighbours of a node u are exactly the nodes contained in a box of volume $\frac{\overline{\deg}}{n}$ around it, i.e. one of side length $2r = \left(\frac{\overline{\deg}}{n}\right)^{1/d}$. Then considering the 1d GIRG whose node locations are the projection of all nodes onto one of the dimensions i, the whole strip of $[0,1]^d$ of volume $2r$ gets projected down into the segment $[(x_u)_i - r, (x_u)_i + r]$. This contains in expectation $n \cdot 2r$ nodes, and is roughly the set of neighbours of u in the ith 1d GIRG. Putting this together means that average degree in the 1d GIRG is $\widetilde{\deg} = 2nr = \Theta(n^{1-1/d})$.

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 is still better than the naive $O(n^2)$ algorithm, and is actually faster than "exponential in d, linear in n" $O(e^{kd}n)$ in the regime where $d \gg \log n$.

Extension to mixed GIRGs Although the algorithm doesn't have as good a runtime on max GIRGs, it's easy to see that it does extend well to a mixed GIRG whose distance function has an outer minimum - of the form $r(x, y) = \min\left(\|x_{I_1} - y_{I_1}\|_1, ..., \|x_{I_k} - y_{I_k}\|_k\right)$. We just need a routine to sample the GIRGs in each dimension subset which also returns the distance in that subset for each edge. Then we can combine each of these GIRGs' edges together to form the final output GIRG. The number of overcounted (and hence needed to be discarded) edges from the subset GIRGs won't exceed the final edge count by more than a factor of d. This depends on the point locations - e.g. in the simple $d = 2$ MCD GIRG case we expect little overcounting, however if the points all lie close to a linear subspace $x_1 = x_2$, then we get almost exactly double counting.

Therefore we can efficiently sample e.g. high dimensional mixed min/max GIRGs as long as each inner max dimension subset is small, such as $|I_j| \leq 3$.

Chapter 3

Generative Graph Models (GGM) and Blasius Realism Framework

Contents

[Bläsius et al., 2018] 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 classifier of graph $\rightarrow$fake/real

11

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

[Bläsius et al., 2018] found poor realism of the tested models on the ~ 100 strong subset of the real-world graphs that were Facebook social networks. We however will use the framework to specifically test different types of GIRGs' realsim on this tough subset, with the hope that they will do better than the best models of [Bläsius et al., 2018] - Chung-Lu and Hypebolic. We also think that restricting to this one subset of what should be similar types of real-world graphs will more clearly distinguish our different GIRG types.

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. We write this as $P(u \sim v) = p$.

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 $\arg\min_p \sum_{u \neq v} \log(p^{e_{uv}}) + \log((1 - p)^{1 - e_{uv}})$, 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:

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

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. 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 $f(G') \cong f(G)$, 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.

Having fit $\hat{\theta}$ of the GGM, we can then sample graphs $G' \sim \mathscr{G}(\hat{\theta})$ which are hypothetically similar to G. We can only hope for similarity on the global level,

as even in the best case scenario where both $G, G' \sim \mathscr{G}(\hat{\theta})$, due to the inherent randomness of the GGM, we cannot expect individual edges to matchup e.g. $e \in E \iff e \in E'$, let alone to be able to identify nodes in G with those in G' - see chapter 6 for more on graph similarity.

For example in the Erdos-Renyi exact fit case, in the best case scenario we have $G \sim \mathscr{G}_{ER}(n, p^*)$ and $G' \sim \mathscr{G}_{ER}(n, \hat{p} = p^* + \varepsilon)$. 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 close to that of the real graph, $E_{G' \sim \mathscr{G}(\hat{\theta})}[f^{\text{class}}(G')] \approxeq f^{\text{class}}(G)$, 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 $\mathscr{G}(\hat{\theta})$, assuming the matching GGM hypothesis, we would not expect G to stand out from the crowd. Essentially $f^{\text{class}}(G'(G)) \sim f^{\text{class}}$ should follow a distribution with $f^{\text{class}}(G)$ close to its median/mean.

3.1.3 Fitted GGMs as candidates for real graphs - Blasius

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

They again fit individual $\hat{\theta}_i$ with which to generate one $G'_i \sim \mathscr{G}(\hat{\theta}_i)$ 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 $f^{\text{class}}(G)$ 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, average node degree, centrality, closeness, diameter, clustering coefficient and so on.

Aggregating the classification over the whole dataset of real graphs allows a

We suggested in the previous section that multiple G'_{ij} could be generated per real graph G_i - this would reduce the variance of the whole process. This is also necessary to be able to train a classifier on an individual real graph basis, as you cannot do binary classification on just a pair of feature vectors $f^{\text{class}}(G_i)$ and $f^{\text{class}}(G'_i)$. Blasius instead aggregates the classification over the whole dataset of real graphs, allowing sufficient datapoints to train a classifier while reducing the overall necessary computation (k generated graphs produced instead of km).

This also has the 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.

For the SVM classifier trained on the mirrored real/fake graph dataset, it will have close to 50% accuracy if really $G_i \sim \mathscr{G}(\theta_i^*)$, and higher accuracy if instead $G_i \sim \tilde{\mathscr{G}}(\phi_i^*)$ some alternative GGM $\tilde{\mathscr{G}}$.

3.2 Fitting GIRG GGM to a real graph

3.2.1 Fitting number of nodes n

We follow [Bläsius et al., 2018] which first preprocesses the graph G by shrinking it to its largest connected component, i.e. $G \leftarrow \text{shrinkToGCC}(G)$ before fitting $\mathscr{G}$ 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, which may not collectively fall under the GIRG model. 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 $d(G, G')$, 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 $G' \sim \mathscr{G}$ 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 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`. Using this fit parameter we are enabled to generate new weight sequences $w_u \sim$ powerlaw(τ) for sampling new GIRGs $G' \sim \mathscr{G}_{\text{GIRG}}(\tau, ...)$.

The degree distribution of the graph G is defined as $dd(x) := \frac{|\{v \in V : d(v) = x\}|}{|V|}$, i.e. the fraction of nodes with degree x. For graphs generated by the GIRG model, we saw that the geometry marginalisation property means that $E[d_u] \propto w_u$. Hence if

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).

node weights are distributed following powerlaw(τ), we expect to see the tail of the degree distribution, $dd(x)$ as $x \to n$ to look like a discrete power law distribution $dd(x) \propto x^{-\tau}$.

Sketch of why GIRG power law weights $\to$ 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 binomial$(n-1, \Theta(w_u/n))$, so we'd like to calculate

$$P(d_u = x) = \int P(d_u = x | w_u = w)\Theta(w^{-\tau})\, dw \qquad (3.1)$$

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. $(x - x^{1/2+\varepsilon})^{-\tau} = (x[1 - x^{-1/2+\varepsilon}])^{-\tau} \approx x^{-\tau}$ for large x. Hence $dd(x) \overset{n\to\infty}{\to} \Theta(x^{-\tau})$ for large x.
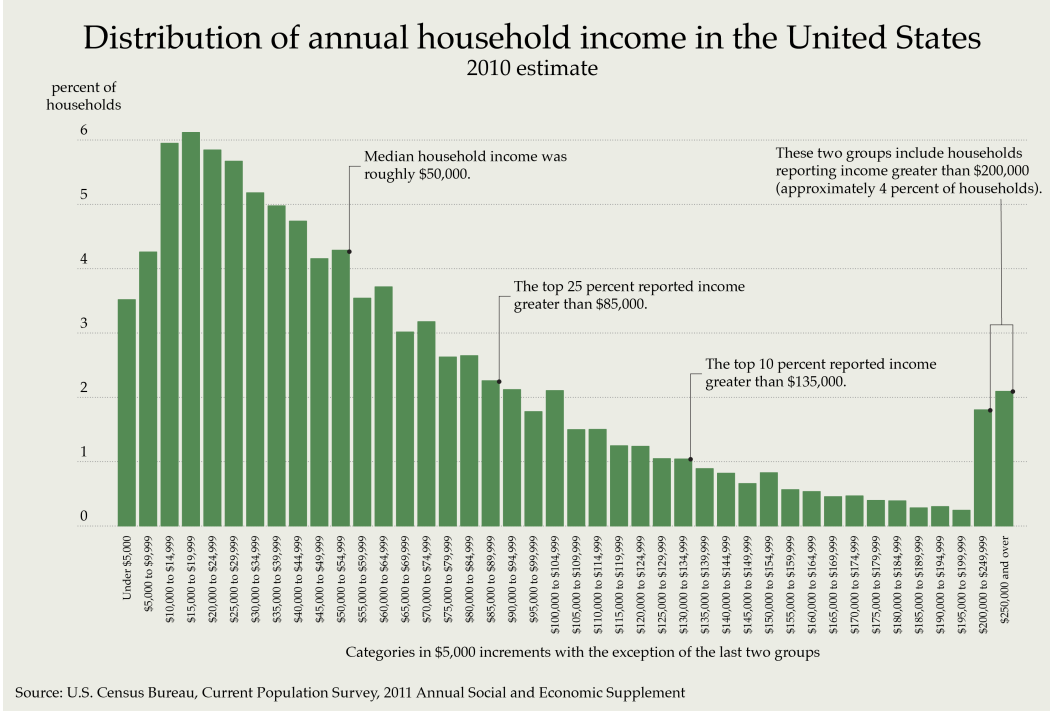
Figure 3.1: Example of a more natural weight distribution with a power law tail but not for smaller weights.

powerlaw package fitting 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. 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 [1]. 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

---

[1]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

well as adding in power law fit Chung-Lu. ER and BA GGMs are still just fit on the courser metric of overall graph average degree.

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 is most directly linked to the generated graph's average degree, whereas the parameter α is more linked to the power of the geometry - larger α decreases the probability of longer distance edges which have $\rho_{uv} < 1$, leaving edges more dominated by shorter (and hence more geometric and clustered) edges. We therefore follow [Bläsius et al., 2018] 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,

$$\text{LCC}(G) := \frac{1}{|V|} \sum_{u \in V} \frac{|\{v, v' \in \Gamma(u) : v \sim v'\}|}{\binom{|\Gamma(u)|}{2}} \tag{3.2}$$

instead of the global clustering coefficient as in [Bläsius et al., 2018]. 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 using binary search, for which we generate a graph $G' \sim \mathscr{G}_{\text{GIRG}}(n, d, \hat{c}_i, \alpha_i, \tau)$, and use the distance metric $\delta(G, G') = |\text{LCC}(G) - \text{LCC}(G')|$ to both propose the next candidate α_i and eventually accept the final best $\hat{\alpha}_i$. Expected LCC is monotone in α so we know which direction to make a new proposal - we follow [Bläsius et al., 2018] in doing binary search on $\alpha^{-1} \in (0, 1)$.

[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 $G' \sim \mathscr{G}_{\text{GIRG}}$. They derive a formula for the expected average degree $\mathbb{E}[\overline{deg}(G')]$ of the GIRG which looks like, emphasizing the dependence on c,

$$\mathbb{E}[\overline{deg}(G')|c] = cA + c^{1/\alpha}B \tag{3.3}$$

Hence we just have to numerically solve eq. (3.3) for $\hat{c} : \mathbb{E}[\overline{deg}(G')|\hat{c}] = \overline{deg}(G) = 2|E|/|V|$ in order to obtain our desired average degree.

The expected average degree formula of eq. (3.3) 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

We'll calculate $E_r[p_{uv}(r)|c]$ for a general volume function $Vol(r)$, and fixed weight sequence $(w_u)_{u \in V}$ and use this to recover eq. (3.3) via

$$E[\overline{deg}(G')|c] = \frac{1}{n} \sum_{u \in V} \sum_{v \in V} E_r[p_{uv}(r)] \tag{3.4}$$

Now $E_r[p_{uv}(r)] = \int_r p_{uv}(r)p(r)dr$. We'll break this down into $\int_{r:\rho_{uv}\geq 1} + \int_{r:\rho_{uv}<1}$ 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 $E_r[p_{uv}(r)] = \int_0^{\hat{r}} p(r)dr + \int_{\hat{r}}^{r_{max}} p_{uv}(r)p(r)dr$.

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

$$E_r[p_{uv}(r)] = \int_0^{Vol(\hat{r})} dVol + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} p_{uv}(Vol)dVol \tag{3.5}$$

$$= Vol(\hat{r}) + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} c \left(\frac{w_u w_v}{W}\right)^\alpha Vol^{-\alpha} dVol \tag{3.6}$$

$$= Vol(\hat{r}) + c \left(\frac{w_u w_v}{W}\right)^\alpha \frac{1}{\alpha-1} \left[\left(\frac{w_u w_v}{W}\right)^{1-\alpha} c^{\frac{1-\alpha}{\alpha}} - 1\right] \tag{3.7}$$

$$= c^{1/\alpha} \left(\frac{w_u w_v}{W}\right) \left[1 + \frac{1}{\alpha-1}\right] - \frac{c}{\alpha-1} \left(\frac{w_u w_v}{W}\right)^\alpha \tag{3.8}$$

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 pairs 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 $c_0 : f(c_0) = \overline{deg}$, and iteratively updating by each time generating a graph $G_i \sim GIRG(c_i)$ 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 around 100 socfb Facebook graphs, whose sizes range from 1000 to 100,000 nodes. For each GGM we produce 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 dataset for a GGM could be e.g. 100% if the fake graphs are highly unrealsitic, or as good as 50% if they're indistinguishable from the real graphs. In fig. 3.5 and fig. 3.7 the numbers displayed are actually the *Misclassifcation rate*, i.e. $1 -$ accuracy, so that higher is more distinguishable.

We present in fig. 3.5 the harder classification test of using a whole distribution of node level features - i.e. for one node level property "deg" would actually include 5 features of the distribution of node degrees: mean, median, lower quartile, upper quartile and standard deviation. In contrast the classification results in fig. 3.7 use just the mean node level features, for which it is easier to fool the classifier. Having more significantly higher than zero misclassification rate numbers in fig. 3.7 allows more comparison between different GGMs and across a wider range of features.

Types of GIRG tested in classification framework

- The "base" GIRG is max norm torus GIRGs for 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 $\|x - y\| = \min \{|x_1 - y_1|_C, \max(|x_2 - y_2|_C, |x_3 - y_3|_C)\}$

- Max norm cube GIRGs - though we were held back due to the increased computation when fitting the average degree with constant c. Hence we just had $d = 1, 2, 3$ cube GIRGs, and $d = 1, 2, 3, 4, 5$ copy-weight cube GIRGs (an attempt for maximal realism)

- The host of simpler models CL, BA, ER as in [Bläsius et al., 2018], as well as Hyperbolic Random Graphs which are very similar to 1d torus GIRGs

New features We additionally added features that weren't assessed in [Bläsius et al., 2018]. Shown in fig. 3.5 are the non node-level "k-cores, comms". These denote k-core sizes and community sizes (communities found using the Louvain method). However they can be rather high variance numbers, for example a 3000 node 2d max torus GIRG with average degree 60 can end up with 40 k-cores and just 8 communities. This means that the distribution features of community size 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 rest, so this feature seems useless.

3.3.1 Degree distribution features

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

For this group we observe much better performance for copy-weight GIRGs / copy-weight Chung-Lu. For all these groups except "n, m, Katz" in fig. 3.5 (distribution features), non copy-weight GGMs have all zero misclassification rates.

They also feature monotonically increasing rates with increasing GIRG dimension, as this moves the graphs to be closer to the Chung-Lu degree distribution, see [Friedrich et al., 2023].

For just mean features there are some 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%) and decently better on Katz centrality (17%) than the other GIRG types. This could be due to a more realism of extreme nodes near the edge of the cube that both have a lowered degree, and whose neighbours are also similarly low degree and extreme. The copy weight GGMs do get up to 47% on Katz, however oddly copy weight cube GIRGs have just $7 - 9\%$ misclassification rate.

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 reality in comparison to the non geometric GGMs, and we can also compare between the different max / MCD / mixed; Torus / Cube GIRG types. We focus solely 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 harder simulation task.

Chung-Lu graphs are a good comparison point against GIRGs by being a similarly inhomogenous graph but without any geometry. [2].

Comparing between 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 most strong 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. 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 for some reason; cube GIRGs are a little worse. However MCD and mixed GIRGs do better on closeness than max GIRGs, especially 1-234 mixed GIRGs. This is actually evidence for the "one strong dimension" hypothesis, as 1-234 mixed GIRGs give more significance to their dim 1 over dims 2,3,4.

Effective diameter and cube GIRGs Cube GIRGs generally underperform, except on effective diameter, where they shine through as the best, with 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 have torus shortcutted. We can dive into see that our intuition is correct: real effective diameters of FB graphs range from 3.0 to 4.4; 2d

---

[2]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.)
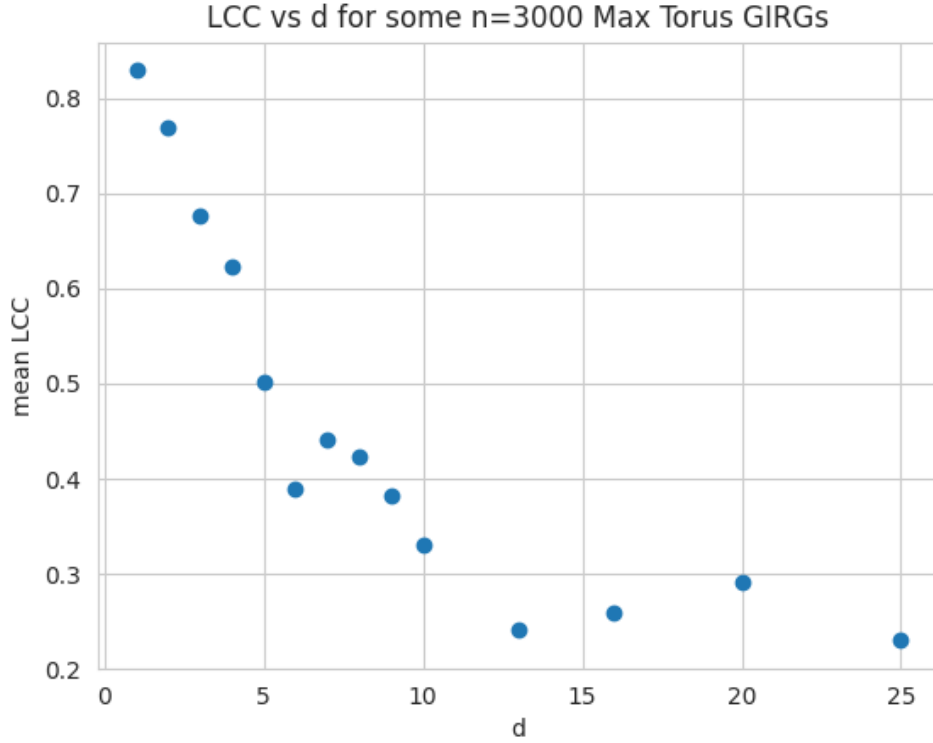
Figure 3.2: Increasing the dimension d of a GIRG will decrease the mean LCC, but only up to a $\Theta(1)$ threshold due to the triangle inequality. Each point is one generated GIRG, all other parameters remaining fixed.

torus GIRGs are too small, on average 0.17 less than their real counterparts, while 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 0% misclassification rate as they have tiny mean LCC. This drops to 0% for all GIRGs except copy-weight cube GIRGs when classifying on LCC distribution. [Bläsius et al., 2018] explained this by finding their fit hyperbolic graphs to have too low variance, which is likely the case here too.

For mean LCC only 4d, 5d MCD GIRGs and 6d, 7d max GIRGs have low misclassification rate, due to their inability to match the real graphs' high mean LCC. E.g. on socfb-Caltech36, the real graph has mean LCC of 0.41, however capping at $\alpha = 100.0$ ($\alpha = \infty$ wouldn't improve the situation much either), the max norm 4d GIRG can only reach 0.33, and the 5d GIRG just 0.24. LCC is guaranteed to be non-tiny 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 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 accuracy: closeness centrality

In fig. 3.5 which provides feature distributions as input to the SVM classifier, it's disappointing that non copy-weight GIRGs have 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% and 2% on "n, m, betw" and "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 doubts.

fig. 3.3a shows that mean closeness fits 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 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.

Unfortunately classification accuracy is a crude metric. One drawback 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 tiny 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.

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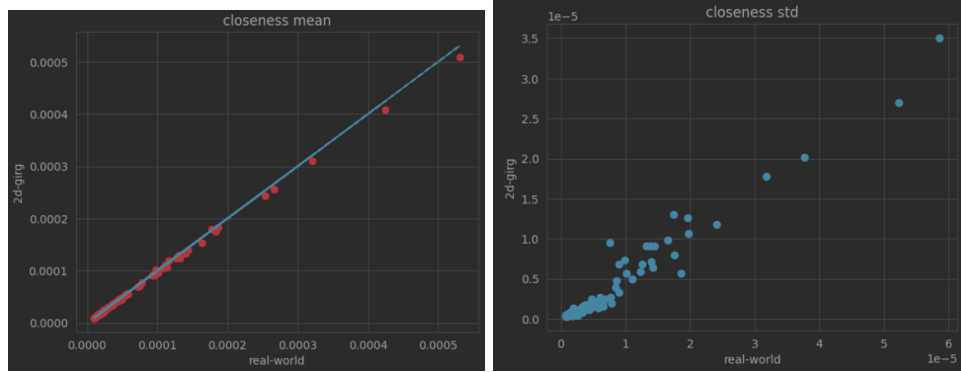
(a) 2d GIRGs have almost identical mean node closeness to real-world FB graphs; $y = x$ line shown in blue.

(b) 2d GIRGs have consistently lower standard deviation of node closeness than real-world FB graphs
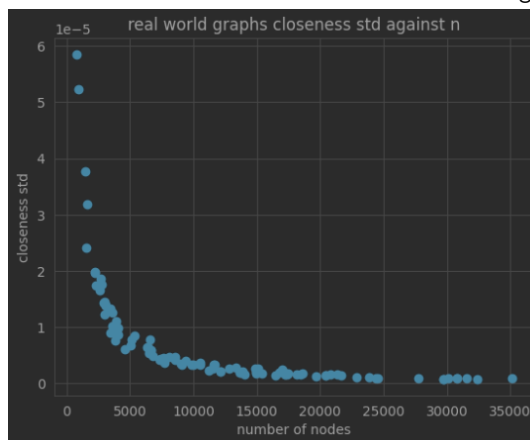
(c) standard deviation of node closeness in real-world FB graphs fits well as a function of number of nodes

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.
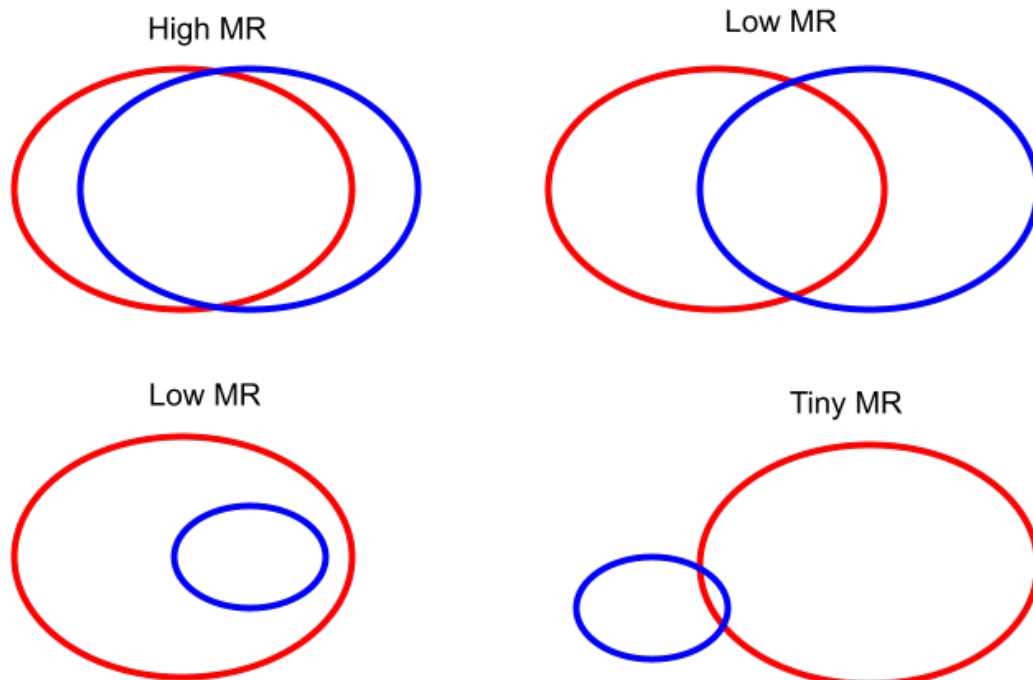

Figure 3.4: Illustration of SVM misclassification rates (MR) given different positions and variance of binary clusters

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., 2018] GGM comparison framework on Facebook graphs, extended to different GIRG models. Some models are missing from the table e.g. 3d-6d GIRGs as their results follow a trend from low to high dimension.

1-ccu	1d copy-weight cube GIRG		betw	betweenness centrality
1-23	1 ∨ (2 ∧ 3) mixed min/max GIRG		k-core	node k-core number
2-min	1 ∨ 2 2d min GIRG		close	closeness centrality
3-cu	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	graph effective diameter

Figure 3.6: Graph Generative Model abbreviations; feature name abbreviations

Features	1ccu	2ccu	3ccu	4ccu	5ccu	CL-c	1-23	1-234	12-34	1-2-34	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	3	1	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	5	3	1	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	50	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	50	49	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	48	47	46	46	0	49	50	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	48	48	50

Figure 3.7: Blasius Framework on just mean values

Chapter 4

Diffusion Maps

Contents

4.1 Introduction

Previous sections' GGM comparison framework by binary classification was designed to compare the similarity of two distributions of graphs based on higher level features. These features like mean/median/stdev node degree / node closeness centrality, or effective diameter etc. are 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. For the GIRG GGM, this means not just fitting $\hat{\alpha}$ 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 $x_u \in \chi$.

We saw this already to some extent with the copy-weight GIRGs - where $\hat{w}_u = d_u$ is fit, as the observed real graph node degrees. The x_u are much harder to fit - for a start any maximum likelihood fit $\hat{x}_u$ 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 fitting an initial good guess of the $\{x_u\}_{u \in V}$ positions.

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 in a high dimensional space, which fall on a lower dimensional manifold - e.g. points conforming to a 3d sphere but embedded in $\mathbb{R}^5$. The method involves first converting points to a (weighted) graph via a distance kernel, and then deconstructing a diffusion process on this graph to produce a lower dimensional vector representation of the original points.

We use the second half of this method, graph $\rightarrow$ lower dim points as a computationally efficient method to discover the underlying geometry of a graph $G = (V, E)$ solely from the connectivity, assuming that it was generated from a d-dimensional GIRG. This allows a good initial guess at the original node locations $(x_u)_{u \in V}$.

4.2.1 Random Walk Formulation

The idea of Diffusion Maps is to analyse the diffusion process of randomly walking on the edges of the graph, and to characterise the probability cloud starting from one node in the graph as a sum of decreasingly important contributions, along the line of eigenvectors of decreasing eigenvalues from 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 diffusion process is defined by the random walk

$$M_{ij} := P(X(t+1) = j | X(t) = i) = \frac{w_{ij}}{\deg(i)} \tag{4.1}$$

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

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

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

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

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

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

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

We use the diagonalisation of S to write M as

$$M = D^{-1/2}SD^{1/2} = \Phi\Lambda\Psi^T \qquad \text{diffusion map representation}$$

$$= \sum_{k=1}^{n} \lambda_k \phi_k \psi_k^T$$

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 $M\mathbf{1} = \mathbf{1}$ since $\sum_j \frac{w_{ij}}{\deg(i)} = 1$, which can be shown to be the largest eigenvalue $\lambda_1 = 1$; if the graph is connected then also $\lambda_2 < 1$. In particular then $\boldsymbol{\phi}_1 = c\mathbf{1}$.

The diffusion map representation of a node is then

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

The truncated diffusion map summarises the distinguishing features of a node's random walk cloud after t steps, truncating off decaying contributions which are scaled by smaller λ_i, Since $\boldsymbol{\phi}_1 = c\mathbf{1}$, 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 $\pi_i = \frac{d_i}{\sum_j d_j}$.

4.2.2 Improving Diffusion Map Geometry

We ideally want the diffusion map embedding of a GIRG generated graph to have similar distances to the original geometry: $\|\boldsymbol{\varphi}(u) - \boldsymbol{\varphi}(v)\| \cong \|\boldsymbol{x}_u - \boldsymbol{x}_v\|$.

Long story short, in the conventional manifold $\mathcal{M}$ in high dim space $\to$ lower dim diff map embedding, indeed "diffusion distance" is equal to "geodesic distance". To be more precise, as [Berry and Harlim, 2018] outlines in its background section, for a finite set of vectors $z_i \in \mathcal{M}$, the diffusion walk on their distance kernel graph approximates a heat like diffusion on the whole continuous manifold. The continuous diffusion distance analogue between any two points $x, y \in \mathcal{M}$ then equal, up to a scale factor, to $d_g(x, y)$, the geodesic distance on the manifold, when $d_g(x, y)^2 \ll t \ll 1$. Finally, diffusion distance equals embedding distance $D_t(x, y) = \|\varphi_t(x) - \varphi_t(y)\|$ (now φ_t is a vector of truncated eigenfunctions on the whole manifold $\mathcal{M}$).

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 have edges $u \sim v$ or $u \nsim v$, which give a signal as to whether $d(u, v)^2$ 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.

---

[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.

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 z_i, 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 formed influenced by some geometric principle, in some finite d-dim space, then this is probably the best approach.

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 $W_{ij} = e^{-\|z_i - z_j\|^2 / T}$, for some parameter $T > 0$. 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 $\|z_i - z_j\|^2$ the expected distance between nodes u, v in the HRG - we can do the same for a GIRG: $\mathbb{E}[\|x_u - x_v\|^2]$. This is our best distance guess given our assumption that G is GIRG generated.

If $u \sim v$, then $E[r_{uv}|w_u, w_v, u \sim v] = \Theta\left[\left(\frac{w_u w_v}{n}\right)^{1/d}\right]$, assuming that $\alpha > 1 + \frac{1}{d}$. We can use the node weight estimates by degree of $\hat{w}_u = d_u$ to get distance estimates $\hat{r}_{uv} = \left(\frac{w_u w_v}{n}\right)^{1/d}$. Finally edge weights are $W_{uv} = e^{-\hat{r}_{uv}^2 / T}$, i.e. decaying with predicted distance.

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 - intuitively this edge is likely to be longer in the χ-space and hence less important to 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. is heat diffusion / brownian motion like.

Again we take d_u as the estimated weight $\hat{w}_u$ of node u. In the original naive random walk M on $W = A$, we can consider choosing an edge out of node u to take as a random walk step. 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 $x_v - x_u$ having an angle $\theta \in [0, 2\pi)$ uniformly randomly), the distance we travel is not random. For brownian motion we'd ideally like to take a small bounded length step in a random direction.

We can modify the naive random walk M to prioritize neighbours v with lower degree d_v via

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

$$M \leftarrow D'MD^{-\gamma} \quad \text{(diagonal D' normalises probabilities)}$$

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 $k(d_u) = d_u^{-\gamma}$.

Picking $\gamma = 1$ seems like a good start - in powerlaw weighted GIRGs, it counteracts the tendency for a node's neighbours to have a shifted weight distribution. I.e. in the base $W_{uv} = 1$ 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 $w_v \sim$ powerlaw$(\tau - 1)$; our modified random walk has $w_v \sim$ powerlaw(τ), matching the original node weight distribution.

With no assumptions on the weight distribution, $\gamma = 1$ also means that the random walk's stationary distribution is more like a constant 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 $\tilde{d}_u = \sum_{v \sim u} d_u^{-1} d_v^{-1} = d_u^{-1} \sum_{v \sim u} d_v^{-1}$. For graphs where most nodes have a decent number of neighbours spread across lower and higher degree nodes, then $\sum_{v \sim u} d_v^{-1} = \Theta(d_u)$, and so $\tilde{\pi}_u \propto d_u^{-1} \Theta(d_u) = \Theta(1)$.

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

We can also quantify the expected random step distance in χ-space. If the starting state is node u at timestep 0, and we transition to node v in the first hop (necessarily $v \sim u$ is a neighbour), then $E[r_{uv}|w_u, u \to v]$ is the expected distance travelled in χ-space due to our transition.

$$E[r_{uv}|w_u, w_v] = \Theta(w_v^{1/d}) \qquad \text{(For } \alpha > 1 + \tfrac{1}{d}. \text{ Dropped factor of } w_u^{1/d})$$
$$p(w_v|v \sim u) \propto w_v^{1-\tau} \qquad \text{(powerlaw weights)}$$

So for example with the Expected distance heat kernel edge weighting,

$$E[r_{uv}|w_u, u \to v]$$
$$= \int E[r_{uv}|w_u, w_v = w] p(w_v = w|u \to v) dw$$
$$= \int \Theta(w^{1/d}) p(w_v = w|u \to v) dw$$
$$= \int \Theta(w^{1/d}) \Theta(p(u \to v|w_v = w, u \sim v) p(w_v = w|u \sim v)) dw$$
$$= \int \Theta(w^{1/d}) \Theta(e^{-w^{2/d}} w^{1-\tau}) dw$$

Which is bounded, unlike in the original naive approach where

$$E[r_{uv}|w_u, u \to v] = \int \Theta(w^{1/d} w^{1-\tau}) dw$$
$$= \infty \quad \text{For } \tau < 2 + \tfrac{1}{d}$$

And in our $\gamma = 1$ approach homogeneous random walk approach,

$$E[r_{uv}|w_u, u \to v] = \int \Theta(w^{1/d} w^{-1} w^{1-\tau}) dw$$
$$< \infty \quad \text{For all } \tau > 2 \text{ powerlaw GIRGs}$$

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 oddly with $\gamma = 0.9$ not $\gamma = 1.0$.
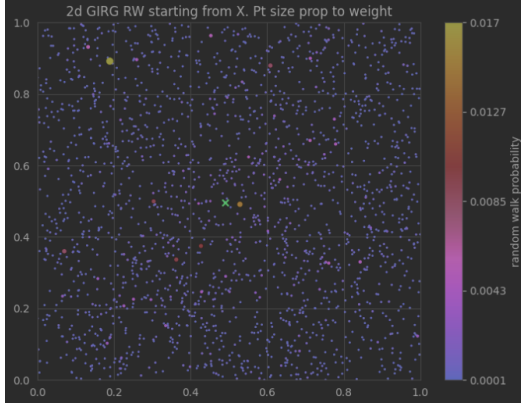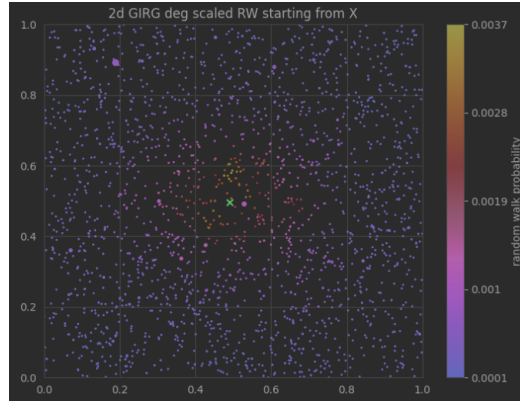
(a) Naive approach: Large weight nodes (drawn with larger radii) get disproportionately more probability mass.
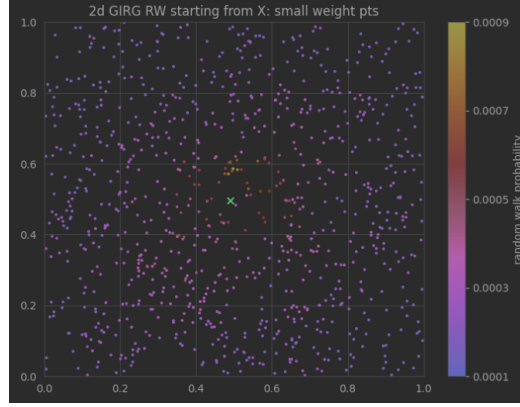
(b) Restriction of previous plot to nodes with smaller weights. Probability distribution looks more gaussian
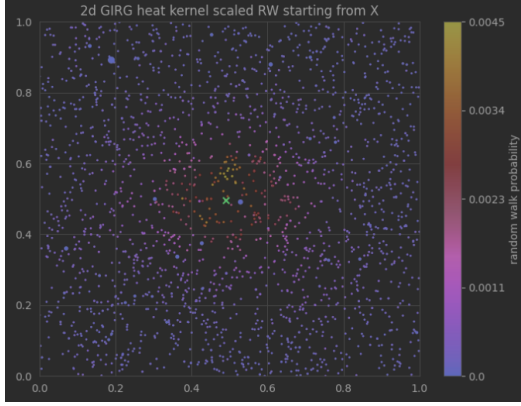
(c) Heat Kernel scaled edge weights using expected distance $W_{uv} = e^{-\hat{r}_{uv}^2}$. Helps to remove the large weight bias - perhaps a little too much.

(d) Scaling transition probabilities by node degree d_i^{-1} also removes the large weight bias.

Figure 4.1: 6 step random walk diffusion cloud starting at the node marked with the green x, for three different (weightings) transition probability schemes. Graph generated from a 2d torus GIRG, whose true point locations give x and y axis.

4.3 Rescaling Points and Empirical Results

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. From section 4.2.2 we know the embedding should be roughly isometric up to a scale factor - so rescaling and shifting into the unit cube/torus is a first good step to make inter-point distances meaningful (even though this could be absorbed into the probability scaling constant c). There are also some other post processing steps that can be done to improve fit for real graphs.

4.3.1 Inferring dimension d

A first question is to choose the output dimension d (truncation for the embedding) of the diffusion map, if the true geometric dimension d of the input graph G is unknown.

(a) $d = 1$

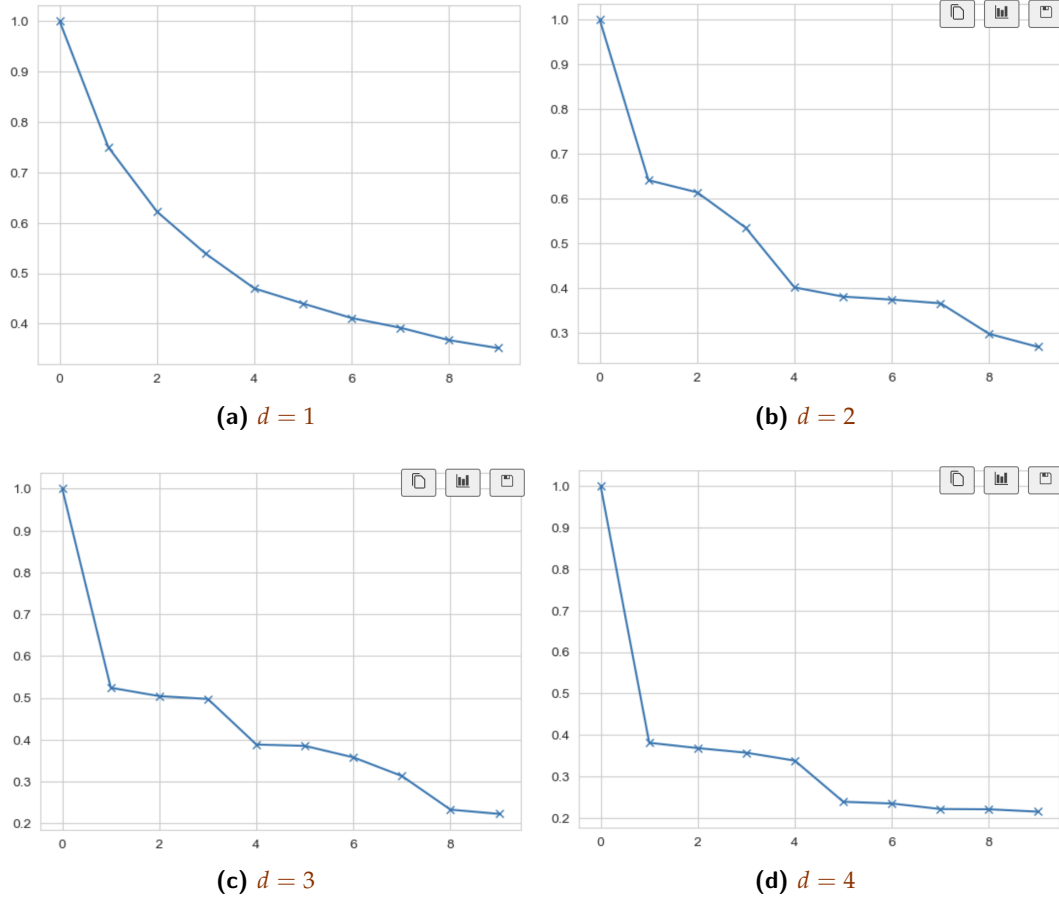**(b)** $d = 2$

(c) $d = 3$

(d) $d = 4$

Figure 4.2: Diffusion map eigenvalues (including $\lambda_1 = 1$) for a $n = 2000, \tau = 2.5, \alpha = 1.3$ Cube GIRG with $d = 1, 2, 3, 4$ dimensions.

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. 4.2 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 - unfortunately messier real world graphs may not be so clear cut.

TODO plot some real graph eigenvalue plots?

4.3.2 Rescaling/Shifting Diffusion Maps

We see in fig. 4.4 some raw $d = 2$ truncated diffusion maps. When the underlying graph was a 2d square GIRG, the inferred points look indeed square like, whereas from a 1d GIRG we do at least get a (curved) line in 2d space.

In general the points will be centered around the origin, as all $\lambda_2, \lambda_3, ...$ eigenvalue eigenvectors will be orthogonal to the stationary distribution $\phi_1 = k\mathbf{1}$: they represent a deviation from the stationary distribution - e.g. for a node on the 1D line towards left end, it will need to put more diffusion probability on the left side nodes, and less on the right side nodes than the stationary distribution. The y-axis scale is small as $\lambda_2 < 1$, and is decreasing with t.
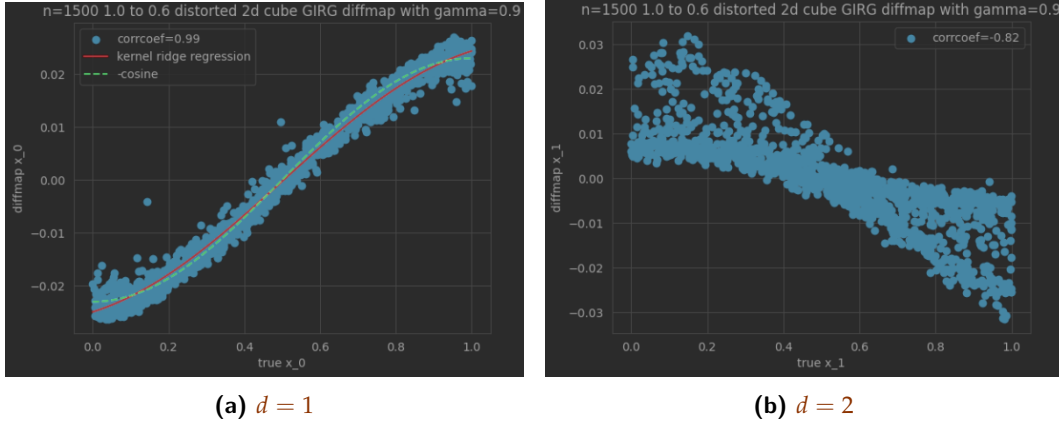
33

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

Figure 4.3: Major and minor diff map against real

The 0 centering can easily be fixed to be more cube GIRG like by shifting the points $(x_u)_i \leftarrow (x_u)_i + \min_v(x_v)_i$. 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 $(x_u)_i$ with its percentile value compared with other $(x_v)_i$. fig. 4.5 shows the "uniformify" procedure in action. This helps to counteract the phenomenon of terribly different diffusion map scaling, whereby the majority of the graph which is well connected ends up highly bunched up, and a small number of outlying poorly connected nodes cover most of the embedding space.

Rotated Points   One issue with diffusion maps as seen in fig. 4.4b, the 2D GIRG's 2D-truncated diff map looks like a rotated square. While the diffusion map has successfully extracted geometric information from the graph, it's not done so in the original basis. This phenomenon can make rescaling/uniformifying to the unit cube a little questionable.

One possible explanation is that as diffusion map is trying to maximise diffusion explainability, the long square diagonal has overall more diffusion along it. This is not very convincing though as there would then be less diffusion in the opposite corners. Even assuming no overall rotation bias, you're going to get a square rotated somewhere between 0 and 45 degrees.

Cuboidal (non-cube) GIRGs   However in practice, real graphs never have an equal balance in geometric dimension importance. This is easiest to understand with the example of a weighted euclidean norm setup, where it could be that a 2D GIRG's true 1st geometric dimension (between $[0,1]$) is more important than its 2nd geometric dimension in influencing edge probabilities: $\|x - y\| = \sqrt{a(x_1 - y_1)^2 + b(x_2 - y_2)^2}$. The diffusion map is solving an eigenvalue problem that is like a maximsation of diffusion explainability. If $a > b$, then by maximisation the first diffusion map coordinate $\varphi_1(u)$ is likely to be very similar to $(x_u)_1$. Only if $a = b$ is there no preference between $(x_u)_1$ and $(x_u)_2$, 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.
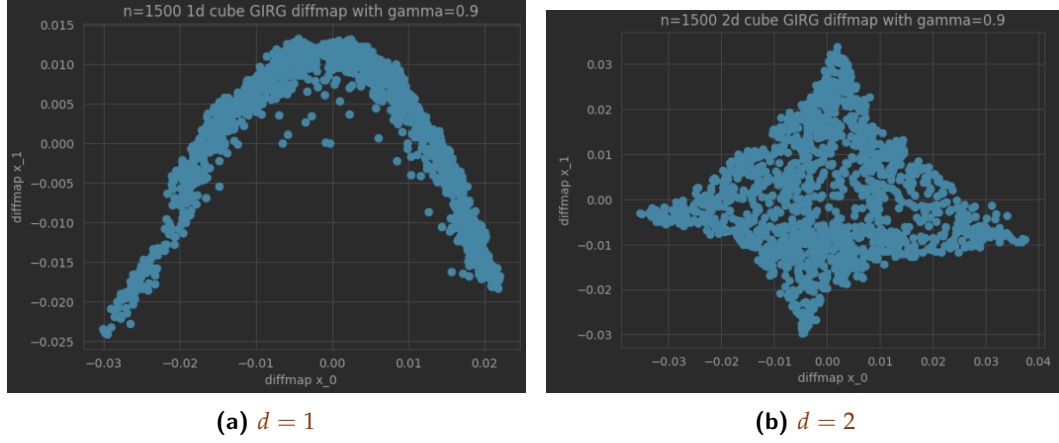
(a) $d = 1$ **(b)** $d = 2$

Figure 4.4: Diffusion map scatter plot of the first two extracted coordinates from 1d and 2d GIRGs.

fig. 4.3a 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 doesn't 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 sine 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. 4.2d as a testament to the underlying cube (non cuboid) GIRG that generated the graph.

Toroidal GIRGs 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 larger eigenvalues λ_2, λ_3 are necessary.

Restricted Rescaling This method works to bias less towards a uniformly distributed prior while still mapping points into a reasonable geometric space. Empirically, the diffusion map coordinates of the 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 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 a uniformify procedure hammer, we will lose the more subtle 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 $[5\%, 95\%]^d$. These nodes 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. 5.4
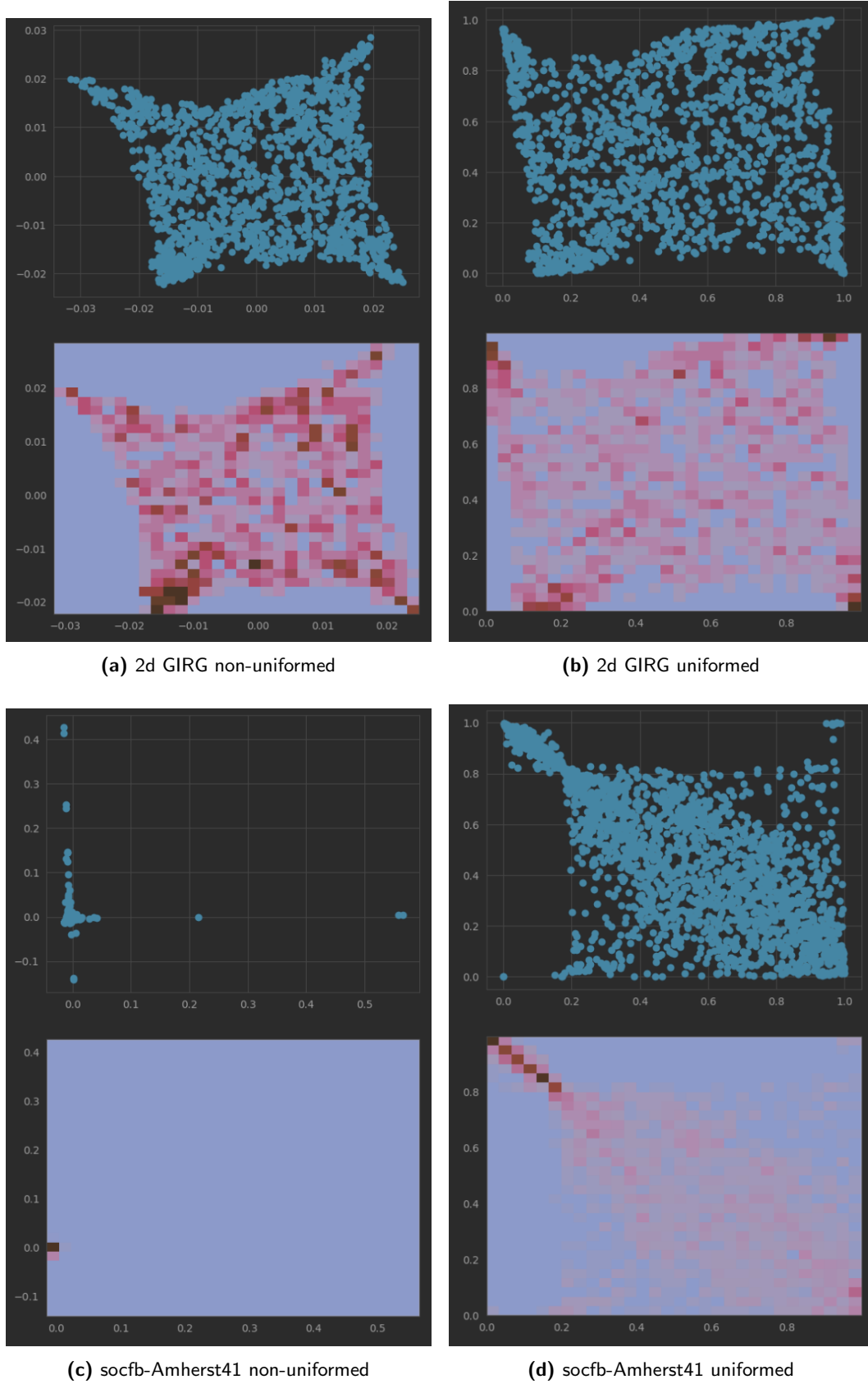
35

(a) 2d GIRG non-uniformed

(b) 2d GIRG uniformed

(c) socfb-Amherst41 non-uniformed

(d) socfb-Amherst41 uniformed

Figure 4.5: Diffusion map scatter plot of the first two extracted coordinates, with and without using an additional uniform square remapping
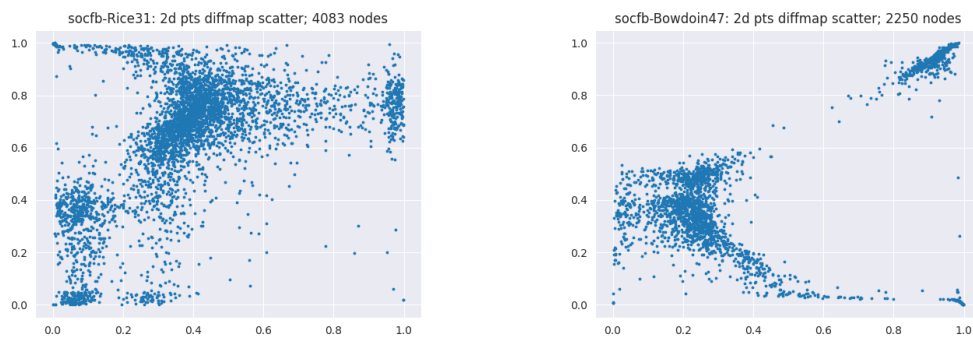
Figure 4.6: the diffmaps (here 2d, restricted and outer uniformified) don't have much point clustering - it's more higher diffusion level than caring so much about individual edges like in the converged case

TODO

- fuller analysis of different diffmap modes: 'uniformify', 'cubify' and 'cuboidify', comparing performance on real life and synthetic graphs

- presentation of the small degree stochastic walk tweak which improves diffusion map performance:

```
# Empirically this gamma seems to work well.
# It discourages taking edges to popular nhbs.
gamma = 0.9
M_tilde = scipy.sparse.diags(1 / D) @ A @ scipy.sparse.diags(D ** (-gamma))
M_tilde = scipy.sparse.diags(np.array(1 / M_tilde.sum(axis=-1)).squeeze()) @ M_tild
```

- is it true that diffusion map tends to cluster representations edges rather than being more uniform? Emprically seems to happen in 2D but not 1D?

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. The actual data likelihood, $p(G|\theta)$, however, is what we'd truly like to maximise. As in [García-Pérez et al., 2019] we can use the likelihood to do a further refinement of point locations, although finding anything like a global optimum is infeasible for all but tiny graphs.

We first tried a Markov Chain Monte Carlo (MCMC) approach to not just improve $p(G|\theta)$ but also, in theory provide an estimate of the posterior distribution of $p(\theta|G)$. While effective, it was quite slow, so we later shifted to using a direct $p(G|\theta)$ optimisation with rounds of point updates, in order from highest to lowest degree node, inspired by [García-Pérez et al., 2019].

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 our node specific fitting GIRG GGM, $\theta = (\alpha, c, \{w_u\}_{u \in V}, \{x_u\}_{u \in V})$, and we focus in particular on the locations x_u. z for us is one real graph instance G.

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. With a proper MC state transition probability $p(\theta \to \theta')$, we can perform 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 $\theta^{j_1}, \theta^{j_2}, \dots$ 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 Gibb's sampling approach. This means breaking down θ into subcomponents θ_i, randomly choosing one to propose a new state for, i.e. $\theta' = (\theta_i', \theta_{-i})$, and using the $Q_i(\theta_i)$ instead of $Q(\theta)$ - i.e. just the marginal non-normalised posterior. In our case $Q(\theta) = p(G|\theta)p(\theta)$, so the uniform prior $p(\theta)$ can be dropped everywhere as $p(\theta) = 1 \; \forall \theta$, and then $p(G|\theta) = \prod_{(u,v) \in \binom{V}{2}} p(e(u,v)|w_u, w_v, x_u, x_v, \alpha, c, G)$. This lends well to Gibb's sampling - we need concentrate only on $Q_u(\theta_u) = \prod_{v \neq u} p(e(u,v)|\dots)$.

Burn in time can be quite long. With the GIRG model, the natural initialisation for x_u would be to follow the prior $x_u \sim U([0,1]^d)$. 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 stochastically propose either a small local perturbation, or a random jump to anywhere in the cube, with some probability of either (we elected for 70% small perturbation). A random uniform jump is useful to try and find a completely new location for x_u that could suit (hopefully near to its neighbours) - in fact another good proposal would be to randomly choose a neighbour $v \sim u$, and move x_u to a random offset of x_v. A small perturbation $x_u' = x_u + \varepsilon$ is good as assuming x_u has high likelihood, somewhere nearby might have even higher.

Acceptance probability   in the Metropolis-Hasting's algorithm

$$A(x_u', x_u) = \min\left(1, \; \frac{p_{prop}(x_u|x_u')p(G|x_u')p(x_u')}{p_{prop}(x_u'|x_u)p(G|x_u)p(x_u)}\right) \tag{5.1}$$

$$= \min\left(1, \; \frac{p(G|x_u')}{p(G|x_u)}\right) \tag{5.2}$$

$$\text{as } p_{prop} \text{ is symmetric and } p(x) = 1 \; \forall x \tag{5.3}$$

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, Chung-Lu model is a good non-geometric null hypothesis to compare against - i.e. we hope the MCMC fit GIRG model will do better than Chung-Lu. It's also interesting to see how large dimension d GIRG is necessary for a good fit.

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

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

The GIRG model is far too confident about edges, giving $p_{uv} \cong 1$ for small $\|x_u - x_v\|$ and $p_{uv} \cong 0$ 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:

$$p_{uv} = (1 - f) \min \left\{ 1, c \left(\frac{w_u w_v / W}{\|x_u - x_v\|^d} \right)^{\alpha} \right\} \tag{5.4}$$

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 high probability of edge), but for long edges (mistakenly predicted low probablity of edge) we need a baseline edge probability to prevent p_{uv} being too small. A simple fix is to mixin the Chung-Lu model - i.e. let

$$p_{uv} = \eta \, p_{uv}^{CL} + (1 - \eta) \, p_{uv}^{GIRG} \tag{5.5}$$

for $0 \leq \eta \leq 1$. 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 mixin parameter 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 very rare.

The intuitive social network interpretation of Chung-Lu mixin is that it allows for some "random" friendships.

With these two new parameters f, η, the augmented GIRG model does have a higher specific fit likelihood than the Chung-Lu model: $p(G|\hat{\theta}_{GIRG}, \mathscr{G}_{GIRG}) > p(G|\hat{\theta}_{CL}, \mathscr{G}_{CL})$. 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 $f = 0.3, \eta = 0.5$. See fig. 5.1 for some likelihood convergence curves on some example graphs.

5.2.2 Percent Edges Captured Metric Comparison

Another 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
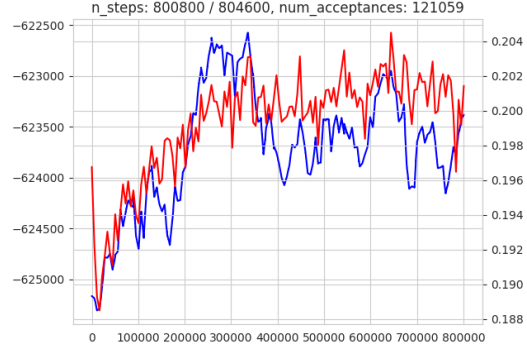
TP	FN
FP	TN

$$\tag{5.6}$$

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.
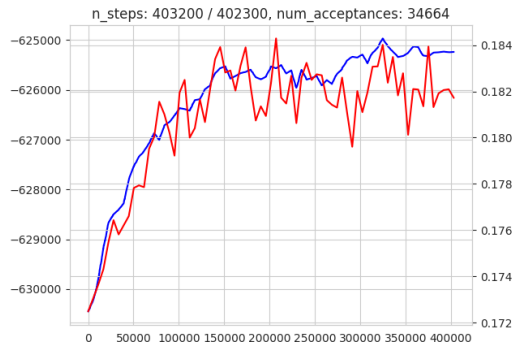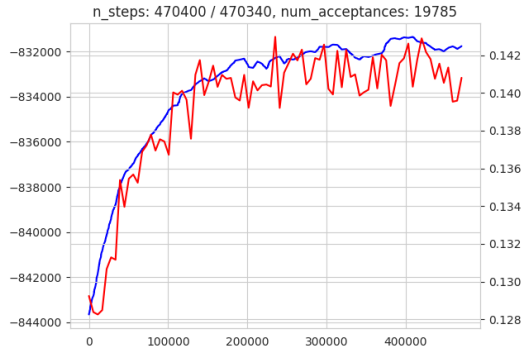
(a) Amherst41 $d = 1$

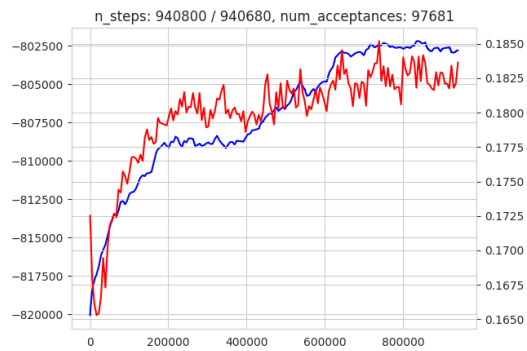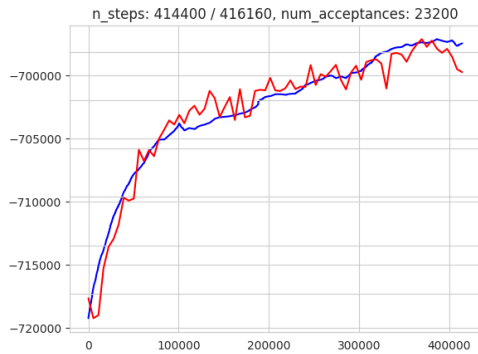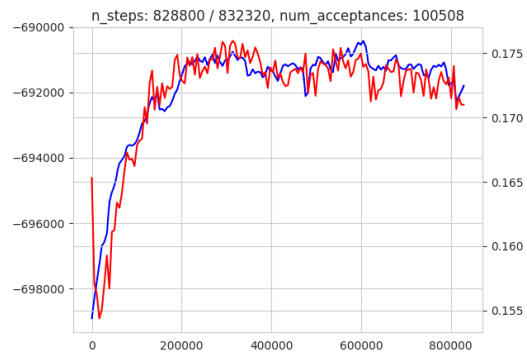**(b)** Amherst41 $d = 2$

(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

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 simpler approach that still bares similarities to MCMC, and inspired by [García-Pérez et al., 2019].

Instead of picking points to update randomly, we sequentially update the positions of every single node, ordered from highest to lowest degree. Updating point x_u is done by proposing 100 new locations - near to current location, near to a neighbour, or random uniform 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 so hence 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.

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. Multiple rounds are repeated until convergence. One optimisation that [García-Pérez et al., 2019] implements but we missed was to also after each round update node weight estimates - comparing two nodes with similar degree, yet one has many geometrically close nodes, and the second has few, the second should have a higher estimated weight.

We see in fig. 5.3 that the converged cube GIRGs can achieve decent PECs. However as the number of nodes in the graph is increased, PEC decreases, e.g. going from about 30% on small graphs of 2000 nodes to 22% on graphs around 10,000 nodes, for just a 1d cube GIRG. However even when the input graph is a GIRG, we still get a similar curve (though overall higher PECs of course). It's possible that full convergence requires more rounds for larger graphs. Not surprisingly for any given graph, $d = 3 > 2 > 1$ in terms of PEC performance.

TODO add in GIRG generated graphs final PEC vs Nodes plot.

Note on the PEC plot: A simple model that might explain what's going on. Say that in the real graph it's 50% GIRG like and 50% random shit. So edge probabilities look like $0.5p_1 + 0.5p_2$, or roughly half edges are from p_1 (GIRG) and half edges are from p_2 (random). We fit a GIRG via $\hat{p}$ to observed edges. We're actually aiming at capturing 100% of the edges, so our we have more prob mass than $0.5p_1$, however let's say that $\hat{p}_1 = p_1 + \Delta p$. Then we fit roughly $0.5 * 0.8$ of GIRG edges and $\bar{d}/2n$ of the remaining edges. So e.g. $y = 0.5 * 0.8 + 0.5 * 60/x$. This is the simple model where GIRG edges are either $p_1 = 0$ or $p_1 = 0.8$.

TODO further improvements might be: updating weight estimates (adding into ordered MCMC, as per mercator). Making code run on GPU.
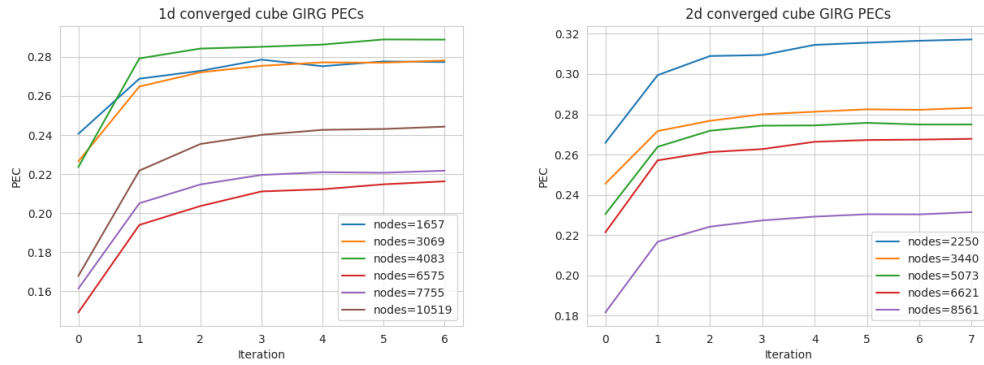
Figure 5.2: Direct ordered likelihood maximisation PEC convergence over iterations for some example socfb graphs.
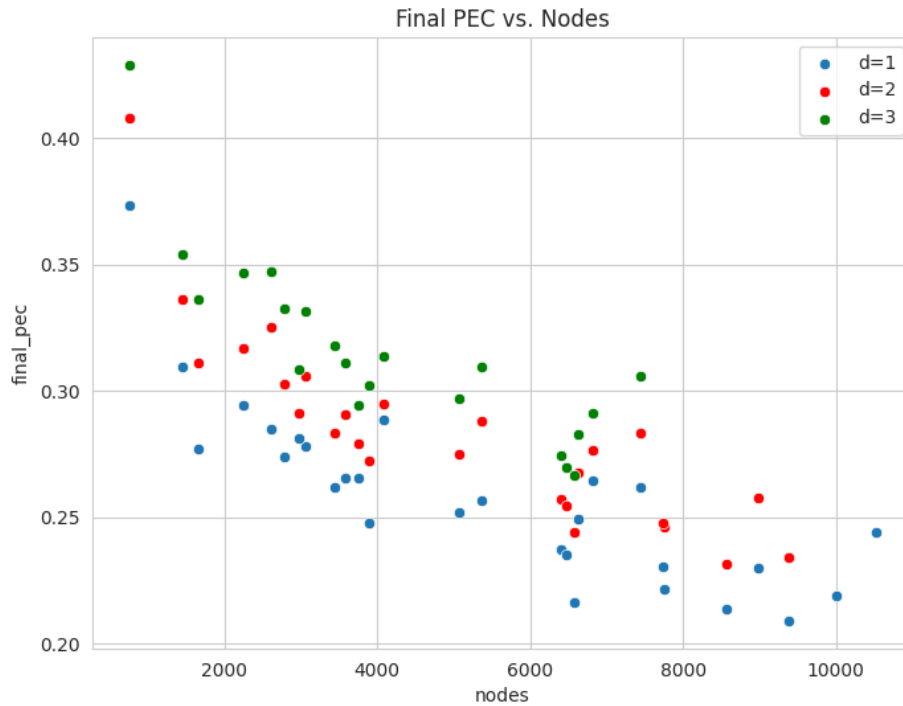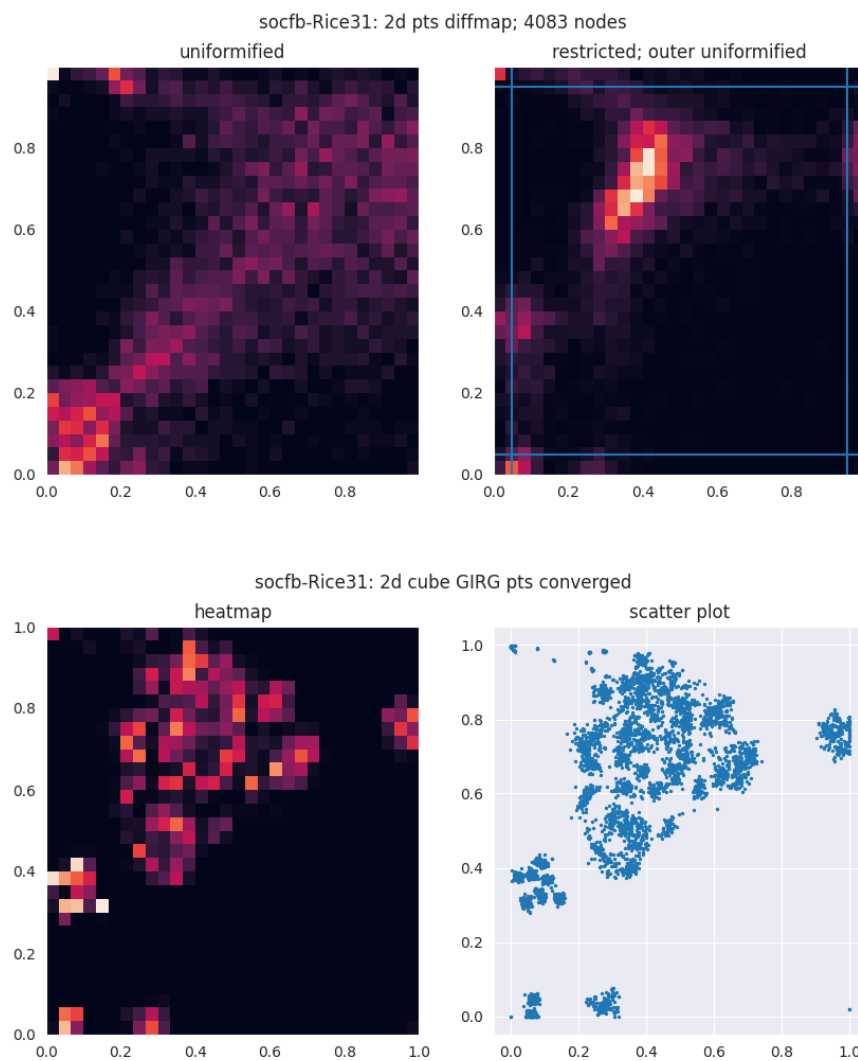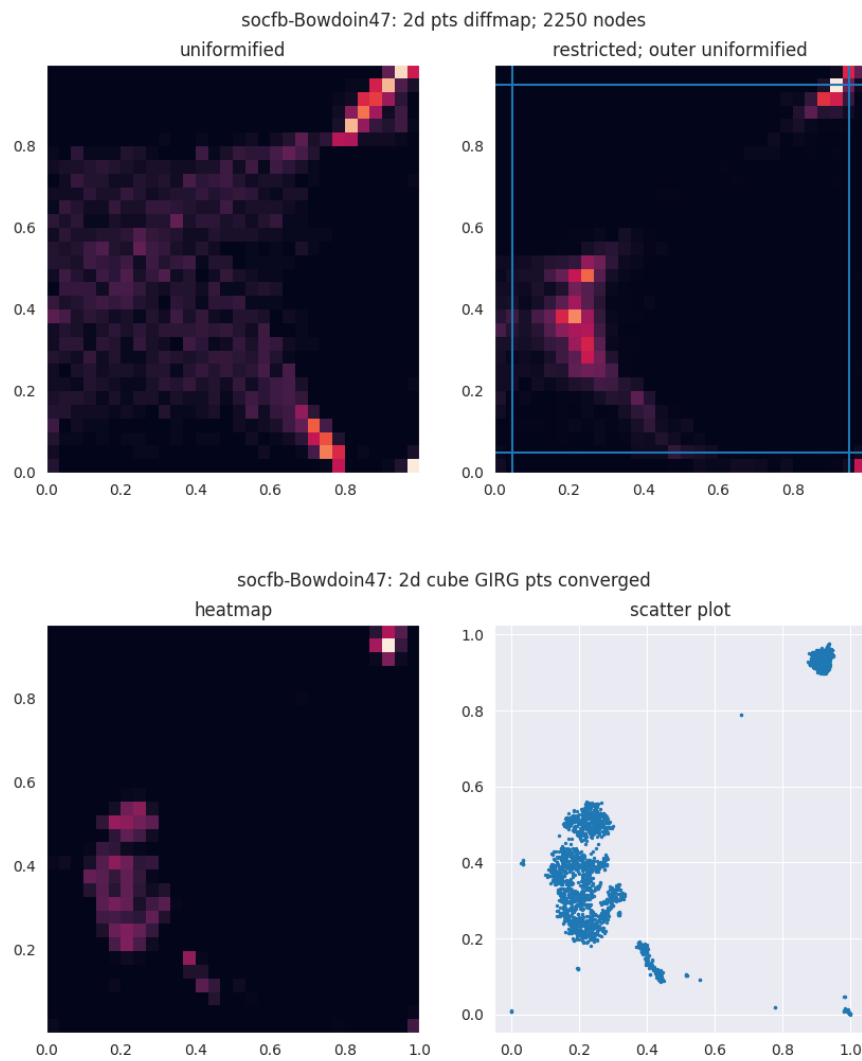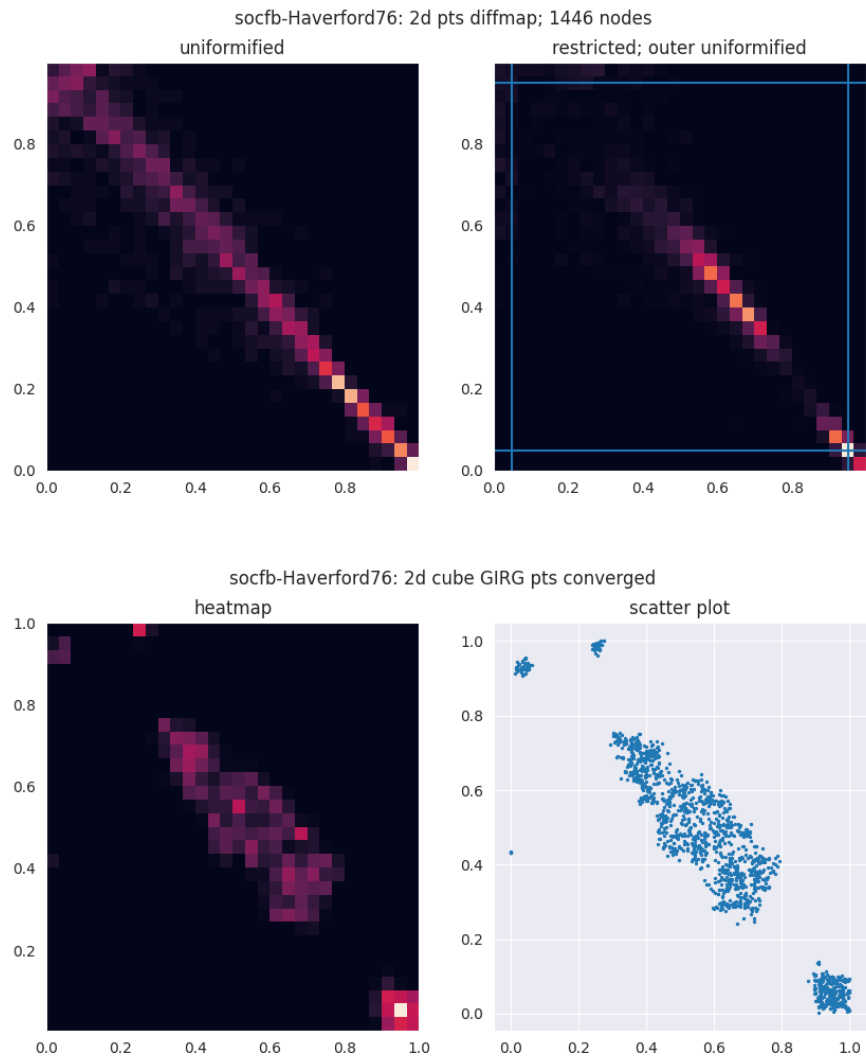

Figure 5.3: converged PECs for 1d, 2d, 3d cube GIRGs. Missing some 3d numbers as batch job exceeded allotted time.

Figure 5.4: 2d cube GIRG fitting examples: Two diffusion map embeddings with different post-processing methods The blue lines for the restricted version show the border at which points are outer uniformified. Then heat map / scatter plot for further refined point locations using direct ordered likelihood maximisation.

socfb-Rice31: 2d pts diffmap; 4083 nodes

socfb-Rice31: 2d cube GIRG pts converged

socfb-Bowdoin47: 2d pts diffmap; 2250 nodes

socfb-Bowdoin47: 2d cube GIRG pts converged

socfb-Haverford76: 2d pts diffmap; 1446 nodes

uniformified — restricted; outer uniformified

socfb-Haverford76: 2d cube GIRG pts converged

heatmap — scatter plot

Chapter 6

Graph Kernels

Contents

In this chapter we try to compare the bayesian evidence based plausibility of different GGMs in generating facebook graphs, with the assistance of graph kernels. Unfortunately the whole endeavour was unsuccessful, but we record our experiments here nonetheless.

6.1 Bayes Factor Theory

A bayesian approach to model selection is to compare $p(M_1|D)$ and $p(M_2|D)$. M_1, M_2 are possible models of the data, e.g. $M_1 = \mathscr{G}_{1D\ GIRG}$ 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 possibly even have a $M_{d=1}, M_{d=2}, M_{d=3}, \dots$ set of models for different dimensional GIRGs with some kind of decaying $p(1d\ GIRG) > p(2d\ GIRG) > p(3d\ GIRG) > \dots$.

For now focussing on just M_1 vs M_2,

$$p(M_1|D) = \frac{p(D|M_1)p(M_1)}{p(D)} \implies \frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)p(M_1)}{p(D|M_2)p(M_2)} \tag{6.1}$$

Hence model selection is done with the ratio $\frac{p(M_1|D)}{p(M_2|D)}$ which is called the **Bayes Factor**. We will ignore the priors $p(M_1), p(M_2)$ for now, and focus on the likelihoods $p(D|M_1), p(D|M_2)$.
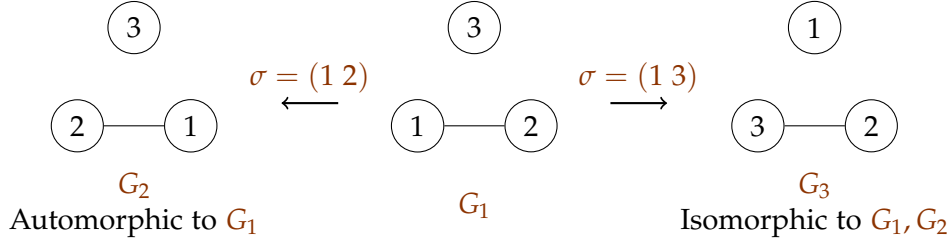
Figure 6.1: Example automorphic/isomorphic graphs

More rigorous bayesian model comparison 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 $\theta = c, \{x_u\}_{u \in V}$, then we could Monte Carlo sample $\theta \sim p(\theta)$ from the prior to get an estimate for

$$p(D|M_1) = p(G|\mathscr{G}_{\text{d-GIRG}}) = \int_\theta p(G|\theta, \mathscr{G}_{\text{d-GIRG}}) p(\theta|\mathscr{G}_{\text{d-GIRG}}) d\theta \qquad (6.2)$$

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. 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 IDs. This is a confusing concept and here an abuse of notation - normally $p(G|\theta)$ is not a sum over permutations, but rather the probability of generating this exact graph. 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.

To be concrete, take an example graph $G = (V, E) = (\{1, 2, 3\}, \{(1, 2)\})$ which is shown as G_1 in fig. 6.1. It's a "labelled graph". G is isomorphic to any other labelled graph H if they look the same if you anonymise the nodes. More formally, if $f : V(G) \to V(H)$ is a bijection mapping nodes to nodes, such that $u \sim v$ in $G \iff f(u) \sim f(v)$ in H, then G is isomorphic to H, and the permutation σ we denoted previously is precisely f. So G is isomorphic to H with edge set $\{(1, 3)\}$. 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 of 3 nodes, there are $3! = 6$ node ID permutations, each of which produces an isomorphic graph G'. For our particular G, we could group together these 6 isomorphic graphs G' into 3 pairs of automorphic graphs.

Hence the bayesian evidence compuation is actually to sample θ, then for each $\sigma \in S_n$ create G' where G is isomorphic to G' with permutation $sigma$ ($G \overset{\sigma}{\to} G'$), and calculate $p(G'|\theta)$; add these up and take the average. Finally repeat for further θ samples and take an outer average.

Consider a simplified Chung-Lu / GIRG model where all nodes have the same weight 1, and $\theta = (x_1, x_2, x_3)$ of the three nodes. The Chung-Lu has identical $p(G')$ for each of the 6 ismorphisms. The GIRG model does not - if x_1, 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.
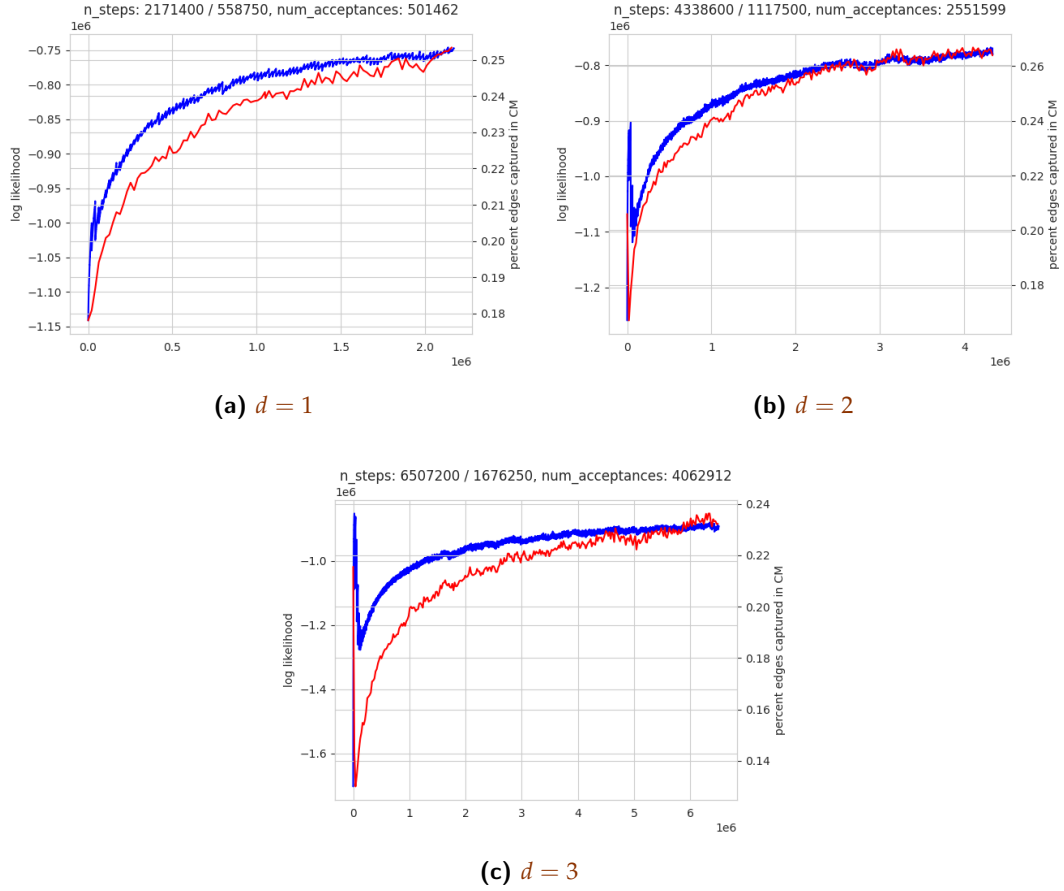
(a) $d = 1$

(b) $d = 2$

(c) $d = 3$

Figure 6.2: MCMC runs for socfb-Amherst41, without failure rate

Unfortunately computing the $n!$ sized $\sum_\sigma p(G' \overset{\sigma}{\cong} G|\theta)p(\sigma)$ 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). Apparently one more computable example is the random walk kernel. Therefore the idea would be to replace the incorrect $p(G|\mathscr{G}_{\text{d-GIRG}}) = \int_\theta p(G|\theta)p(\theta)d\theta$ with $\mu(G) = E_{G'\sim\mathscr{G}_{\text{d-GIRG}}}[k(G, G')]$.

We see in fig. 6.2 that Bayes Factor model comparison wise, 1D GIRGs are superior, even though according to edge accuracy, 2D GIRGs achieve a slight edge.

6.2 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 give a simplified version of Blasius' classification framework. Blasius compares multiple different graph feature combinations on which to train an SVM for distinguishing two graph datasets. Instead we can replace this with a single

kernel which hopefully encapsulates sufficient relevant information on the graph. The question of which graph features to use then shifts to which graph kernel to use!

Another benefit of graph kernels is, as a similarity metric, they give easier direct comparison between graphs. In the previous sections' proposed paradigm, we can directly compare the similarity of a real graph G with two synthetic graphs G_1, G_2, produced from different models M_1, M_2, using $k(G, G_1)$ and $k(G, G_2)$.

6.3 Experiments with Random Walk Kernel; Weisfeiler-Lehman Kernel

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, however unfortunately both proved unsatisfactory. Without further expertise on graph kernels, we were forced to abandon this line of attack. We think that graph kernels might give more meaningful similarity metrics on smaller graphs with more unique structure (our random graph models and the real graphs themselves have a lot of homogenous structure) and particularly meangingful node labels.

6.3.1 Random Walk Kernel

The random walk kernel between two graphs is conceptually a count of the number of random walks of any length l that exist in the product graph. This 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 between G, G' is defined as $G_\times = (V_\times, E_\times)$ where $V_\times = \{(v, v') | v \in V, v' \in V'\}$ and $E_\times = \{((v_1, v_1'), (v_2, v_2')) | v_1 \sim v_2 \in E_1, v_1' \sim v_2' \in E_2\}$. 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.

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. 6.4.

6.3.2 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 the number of iterations of the algorithm. The algorithm requires some kind of node labelling - we colour nodes into a small discreet set of colours by grouping together nodes with similar sized degrees. It also uses a base kernel which is computed at each iteration - we used the simplest/speediest node label histogram dot product kernel: $k(G, G') = \langle f, f' \rangle$ where $f = (f_1, \ldots, f_k)$, $f_i = |\{v \in V : l(v) = i\}|$ is the number of nodes with label i.

The output is $k_{WLK}(G, G') = k_{\text{base}}(G_1, G_1') + \cdots + k_{\text{base}}(G_h, G_h')$. $G = G_1$, $G' = G_1'$, and G_{i+1} is computed iteratively as relabelling each node in G_i with $l(v) \leftarrow (l(v), (l(u))_{u \sim v})$. This looks odd, but in practice each label is just rehashed as a new integer rather than becoming a highly nested tuple.
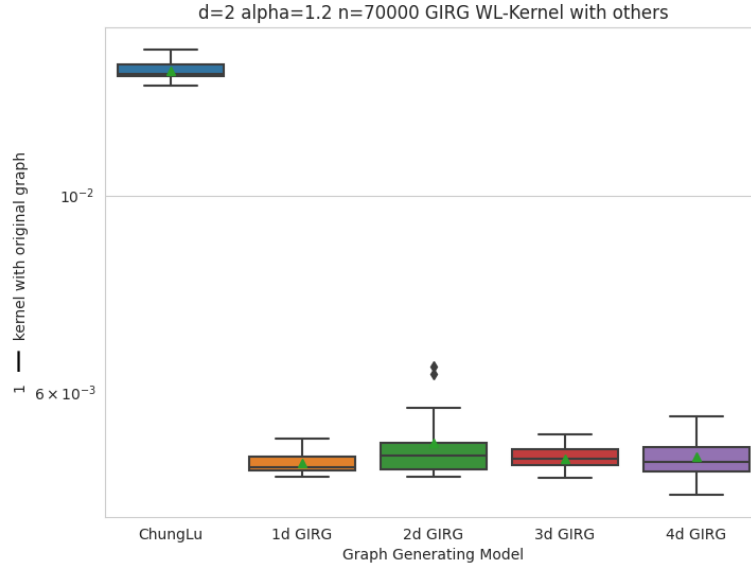
Figure 6.3: WL-Kernel of a d=2, alpha=1.2, n=70000 Torus GIRG with other generated graphs (13 per model). All the GIRGs are more similar to the original than Chung-Lu, but we cannot differentiate between GIRGs of different dimensions.

6.3.3 Experiments

Unfortunately both kernels failed to pass a basic test of ratifying one type of GIRG model over another. We tried a variety of combinations, but didn't get such reliable results: see fig. 6.3 and fig. 6.4.

E.g. in fig. 6.4, the random walk kernel faield the basic test of having $k(G \sim \mathcal{G}_1, G_1' \sim \mathcal{G}_1) > k(G \sim \mathcal{G}_1, G_2' \sim \mathcal{G}_2)$, where $\mathcal{G}_1$ is a 3D copy weight cube GIRG, and $\mathcal{G}_2$ is copy weight Chung-Lu.
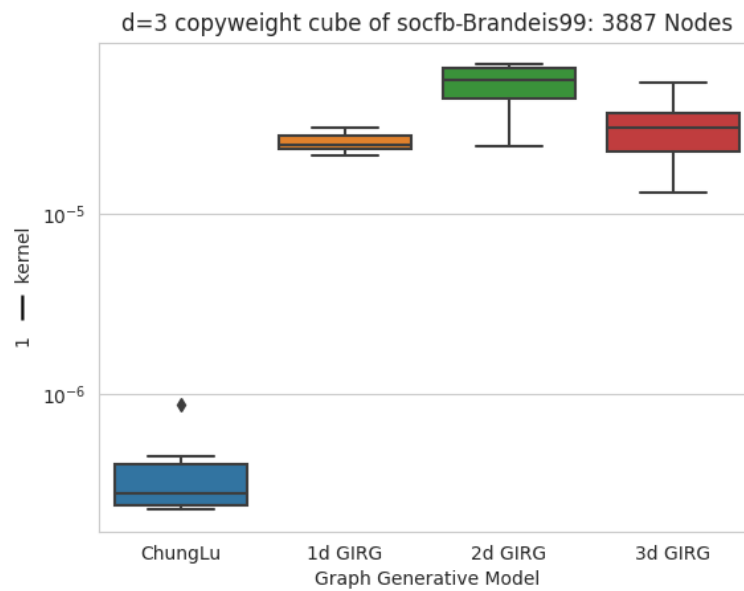
Figure 6.4: RW-Kernel of a d=3 copy weight cube GIRG fit to socfb-Brandeis99 (matching number of edges and local clustering coefficient), with other generated graphs (6 per model type). Chung-Lu graphs have highest similarity to the original, despite it being a 3D GIRG

Chapter 7

Conclusion

Contents

7.1 Recap

The goal of this thesis was to investigate how well the GIRG model can match real graphs with some suspected inherent geometry and power law degree distribution.

Our first attempt was to follow the framework of [Bläsius et al., 2018], to compare a range of generative graph models, including many GIRG subtypes, in their ability to replicate global statistics of a set of 100 social network graphs. In this arena we did find GIRGs able to outperform on a few statistics: closeness centrality, Katz centrality, effective diameter, and average local clustering coefficient.

We then explored fitting GIRG location parameters to these real graphs to see if they're capable of replicating the real graphs on an edge level. This found a decent amount of success.

We tried to use graph kernels in a bayesian framework for assessing evidence of GIRG models against Chung-Lu as generative models of the real graphs. However graph kernels proved incapable of correctly identifying the generative model even when the assessed graph was generated by the same model. Hence we abandoned this avenue.

Appendix A

Dummy Appendix

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

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., 2018] Bläsius, T., Friedrich, T., Katzmann, M., Krohmer, A., and Striebel, J. (2018). 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.

[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.

[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.

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.