ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

This is version `v1.4` of the template.

We assume that you found this template on our institute's website, so we do not repeat everything stated there. Consult the website again for pointers to further reading about LaTeX. This chapter only gives a brief overview of the files you are looking at.

1.1 Features

The rest of this document shows off a few features of the template files. Look at the source code to see which macros we used!

The template is divided into TeX files as follows:

1. `thesis.tex` is the main file.
2. `extrapackages.tex` holds extra package includes.
3. `layoutsetup.tex` defines the style used in this document.
4. `theoremsetup.tex` declares the theorem-like environments.
5. `macrosetup.tex` defines extra macros that you may find useful.
6. `introduction.tex` contains this text.
7. `sections.tex` is a quick demo of each sectioning level available.
8. `refs.bib` is an example bibliography file. You can use BibTeX to quote references. For example, read [Bringhurst, 1996] if you can get a hold of it.

1.1.1 Extra package includes

The file `extrapackages.tex` lists some packages that usually come in handy. Simply have a look at the source code. We have added the following comments based on our experiences:

REC This package is recommended.

OPT This package is optional. It usually solves a specific problem in a clever way.

ADV This package is for the advanced user, but solves a problem frequent enough that we mention it. Consult the package's documentation.

As a small example, here is a reference to the Section *Features* typeset with the recommended *varioref* package:

> See Section section 1.1 on the preceding page.

1.1.2 Layout setup

This defines the overall look of the document aaaaa – for example, it changes the chapter and section heading appearance. We consider this a 'do not touch' area. Take a look at the excellent *Memoir* documentation before changing it.

In fact, take a look at the excellent *Memoir* documentation, full stop.

1.1.3 Theorem setup

This file defines a bunch of theorem-like environments.

Theorem 1.1 *An example theorem.*

Proof Proof text goes here. □

Note that the q.e.d. symbol moves to the correct place automatically if you end the proof with an `enumerate` or `displaymath`. You do not need to use `\qedhere` as with *amsthm*.

Theorem 1.2 (Some Famous Guy) *Another example theorem.*

Proof This proof

1. ends in an enumerate. □

Proposition 1.3 *Note that all theorem-like environments are by default numbered on the same counter.*

Proof This proof ends in a display like so:

$$f(x) = x^2. \qquad \square$$

1.1.4 Macro setup

For now the macro setup only shows how to define some basic macros, and how to use a neat feature of the *mathtools* package:

$$|a|\,, \quad |*|\,\frac{a}{b}, \quad |[|\,]\,\frac{a}{b}.$$

Chapter 2

Writing scientific texts in English

This chapter was originally a separate document written by Reto Spöhel. It is reprinted here so that the template can serve as a quick guide to thesis writing, and to provide some more example material to give you a feeling for good typesetting.

2.1 Basic writing rules

The following rules need little further explanation; they are best understood by looking at the example in the booklet by Knuth et al., §2–§3.

Rule 2.1 Write texts, not chains of formulas.

More specifically, write full sentences that are logically interconnected by phrases like 'Therefore', 'However', 'On the other hand', etc. where appropriate.

Rule 2.2 Displayed formulas should be embedded in your text and punctuated with it.

In other words, your writing should not be divided into 'text parts' and 'formula parts'; instead the formulas should be tied together by your prose such that there is a natural flow to your writing.

2.2 Being nice to the reader

Try to write your text in such a way that a reader enjoys reading it. That's of course a lofty goal, but nevertheless one you should aspire to!

Rule 2.3 Be nice to the reader.

Give some intuition or easy example for definitions and theorems which might be hard to digest. Remind the reader of notations you introduced many pages ago – chances are he has forgotten them. Illustrate your writing with diagrams and pictures where this helps the reader. Etc.

Rule 2.4 Organize your writing.

Think carefully about how you subdivide your thesis into chapters, sections, and possibly subsections. Give overviews at the beginning of your thesis and of each chapter, so the reader knows what to expect. In proofs, outline the main ideas before going into technical details. Give the reader the opportunity to 'catch up with you' by summing up your findings periodically.

Useful phrases: 'So far we have shown that . . . ', 'It remains to show that . . . ', 'Recall that we want to prove inequality (7), as this will allow us to deduce that . . . ', 'Thus we can conclude that . . . . Next, we would like to find out whether . . . ', etc.

Rule 2.5 Don't say the same thing twice without telling the reader that you are saying it twice.

Repetition of key ideas is important and helpful. However, if you present the same idea, definition or observation twice (in the same or different words) without telling the reader, he will be looking for something new where there is nothing new.

Useful phrases: 'Recall that [we have seen in Chapter 5 that] . . . ', 'As argued before / in the proof of Lemma 3, . . . ', 'As mentioned in the introduction, . . . ', 'In other words, . . . ', etc.

Rule 2.6 Don't make statements that you will justify later without telling the reader that you will justify them later.

This rule also applies when the justification is coming right in the next sentence! The reasoning should be clear: if you violate it, the reader will lose valuable time trying to figure out on his own what you were going to explain to him anyway.

Useful phrases: 'Next we argue that . . . ', 'As we shall see, . . . ', 'We will see in the next section that . . . , etc.

2.3 A few important grammar rules

Rule 2.7 There is (almost) *never* a comma before 'that'.

It's really that simple. Examples:

> We assume that . . .
> *Wir nehmen an, dass . . .*

> It follows that . . .
> *Daraus folgt, dass . . .*

> 'thrice' is a word that is seldom used.
> *'thrice' ist ein Wort, das selten verwendet wird.*

Exceptions to this rule are rare and usually pretty obvious. For example, you may end up with a comma before 'that' because 'i.e.' is spelled out as 'that is':

> For $p(n) = \log n/n$ we have . . . However, if we choose p a little bit higher, that is $p(n) = (1+\varepsilon)\log n/n$ for some $\varepsilon > 0$, we obtain that. . .

Or you may get a comma before 'that' because there is some additional information inserted in the middle of your sentence:

> Thus we found a number, namely n_0, that satisfies equation (13).

If the additional information is left out, the sentence has no comma:

> Thus we found a number that satisfies equation (13).

(For 'that' as a relative pronoun, see also Rules 2.9 and 2.10 below.)

Rule 2.8 There is usually no comma before 'if'.

Example:

> A graph is not 3-colorable if it contains a 4-clique.
> *Ein Graph ist nicht 3-färbbar, wenn er eine 4-Clique enthält.*

However, if the 'if' clause comes first, it is usually separated from the main clause by a comma:

> If a graph contains a 4-clique, it is not 3-colorable .
> *Wenn ein Graph eine 4-Clique enthält, ist er nicht 3-färbbar.*

There are more exceptions to these rules than to Rule 2.7, which is why we are not discussing them here. Just keep in mind: don't put a comma before 'if' without good reason.

Rule 2.9 Non-defining relative clauses have commas.

Rule 2.10 Defining relative clauses have no commas.

In English, it is very important to distinguish between two types of relative clauses: defining and non-defining ones. This is a distinction you absolutely need to understand to write scientific texts, because mistakes in this area actually distort the meaning of your text!

It's probably easier to explain first what a *non-defining* relative clause is. A non-defining relative clauses simply gives additional information *that could also be left out* (or given in a separate sentence). For example, the sentence

> The WeirdSort algorithm, which was found by the famous mathematician John Doe, is theoretically best possible but difficult to implement in practice.

would be fully understandable if the relative clause were left out completely. It could also be rephrased as two separate sentences:

> The WeirdSort algorithm is theoretically best possible but difficult to implement in practice. [By the way,] WeirdSort was found by the famous mathematician John Doe.

This is what a non-defining relative clause is. *Non-defining relative clauses are always written with commas.* As a corollary we obtain that you cannot use 'that' in non-defining relative clauses (see Rule 2.7!). It would be wrong to write

> ~~The WeirdSort algorithm, that was found by the famous mathematician John Doe, is theoretically best possible but difficult to implement in practice.~~

A special case that warrants its own example is when 'which' is referring to the entire preceding sentence:

> Thus inequality (7) is true, which implies that the Riemann hypothesis holds.

As before, this is a non-defining relative sentence (it could be left out) and therefore needs a comma.

So let's discuss *defining* relative clauses next. A defining relative clause tells the reader *which specific item the main clause is talking about*. Leaving it out either changes the meaning of the sentence or renders it incomprehensible altogether. Consider the following example:

> The WeirdSort algorithm is difficult to implement in practice. In contrast, the algorithm that we suggest is very simple.

Here the relative clause 'that we suggest' cannot be left out – the remaining sentence would make no sense since the reader would not know which algorithm it is talking about. This is what a defining relative clause is. *Defining relative clauses are never written with commas.* Usually, you can use both 'that' and 'which' in defining relative clauses, although in many cases 'that' sounds better.

As a final example, consider the following sentence:

> For the elements in $\mathcal{B}$ which satisfy property (A), we know that equation (37) holds.

This sentence does not make a statement about all elements in $\mathcal{B}$, only about those satisfying property (A). The relative clause is *defining*. (Thus we could also use 'that' in place of 'which'.)

Table 2.1: Things you (usually) don't say

~~It holds (that) . . .~~	We have . . .	*Es gilt . . .*
('Equation (5) holds.' is fine, though.)		
~~x fulfills property $\mathcal{P}$.~~	x satisfies property $\mathcal{P}$.	*x erfüllt Eigenschaft $\mathcal{P}$.*
~~in average~~	on average	*im Durchschnitt*
~~estimation~~	estimate	*Abschätzung*
~~composed number~~	composite number	*zusammengesetzte Zahl*
~~with the help of~~	using	*mit Hilfe von*
~~surely~~	clearly	*sicher, bestimmt*
~~monotonously increasing~~	monotonically incr.	*monoton steigend*
(Actually, in most cases 'increasing' is just fine.)		

In contrast, if we add a comma the sentence reads

> For the elements in $\mathcal{B}$, which satisfy property (A), we know that equation (37) holds.

Now the relative clause is *non-defining* – it just mentions in passing that all elements in $\mathcal{B}$ satisfy property (A). The main clause states that equation (37) holds for *all* elements in $\mathcal{B}$. See the difference?

2.4 Things you (usually) don't say in English – and what to say instead

Table 2.1 lists some common mistakes and alternatives. The entries should not be taken as gospel – they don't necessarily mean that a given word or formulation is wrong under all circumstances (obviously, this depends a lot on the context). However, in nine out of ten instances the suggested alternative is the better word to use.

Chapter 3

Typography

3.1 Punctuation

Rule 3.1 Use opening (‘) and closing (’) quotation marks correctly.

In LaTeX, the closing quotation mark is typed like a normal apostrophe, while the opening quotation mark is typed using the French *accent grave* on your keyboard (the *accent grave* is the one going down, as in *frère*).

Note that any punctuation that *semantically* follows quoted speech goes inside the quotes in American English, but outside in Britain. Also, Americans use double quotes first. Oppose

> “Using ‘lasers,’ we punch a hole in . . . the Ozone Layer,” Dr. Evil said.

to

> ‘Using “lasers”, we punch a hole in . . . the Ozone Layer’, Dr. Evil said.

Rule 3.2 Use hyphens (-), en-dashes (–) and em-dashes (—) correctly.

A hyphen is only used in words like ‘well-known’, ‘3-colorable’ etc., or to separate words that continue in the next line (which is known as hyphenation). It is entered as a single ASCII hyphen character (`-`).

To denote ranges of numbers, chapters, etc., use an en-dash (entered as two ASCII hyphens `--`) with no spaces on either side. For example, using Equations (1)–(3), we see. . .

As the equivalent of the German *Gedankenstrich*, use an en-dash with spaces on both sides – in the title of Section 2.4, it would be wrong to use a hyphen instead of the dash. (Some English authors use the even longer emdash (—)

instead, which is typed as three subsequent hyphens in LaTeX. This emdash is used without spaces around it—like so.)

3.2 Spacing

Rule 3.3 Do not add spacing manually.

You should never use the commands `\\` (except within tabulars and arrays), `\␣` (except to prevent a sentence-ending space after Dr. and such), `\vspace`, `\hspace`, etc. The choices programmed into LaTeX and this style should cover almost all cases. Doing it manually quickly leads to inconsistent spacing, which looks terrible. Note that this list of commands is by no means conclusive.

Rule 3.4 Judiciously insert spacing in maths where it helps.

This directly contradicts Rule 3.3, but in some cases TeX fails to correctly decide how much spacing is required. For example, consider

$$f(a,b) = f(a+b,a-b).$$

In such cases, inserting a thin math space `\,` greatly increases readability:

$$f(a,b) = f(a+b,\, a-b).$$

Along similar lines, there are variations of some symbols with different spacing. For example, Lagrange's Theorem states that $|G| = [G : H]\,|H|$, but the proof uses a bijection $f\colon aH \to bH$. (Note how the first colon is symmetrically spaced, but the second is not.)

Rule 3.5 Learn when to use `\␣` and `\@`.

Unless you use 'french spacing', the space at the end of a sentence is slightly larger than the normal interword space.

The rule used by TeX is that any space following a period, exclamation mark or question mark is sentence-ending, except for periods preceded by an upper-case letter. Inserting \ before a space turns it into an interword space, and inserting `\@` before a period makes it sentence-ending. This means you should write

```
Prof.\ Dr.\ A. Steger is a member of CADMO\@.
If you want to write a thesis with her, you
should use this template.
```

which turns into

> Prof. Dr. A. Steger is a member of CADMO. If you want to write
> a thesis with her, you should use this template.

The effect becomes more dramatic in lines that are stretched slightly during justification:

> Prof. Dr. A. Steger is a member of CADMO. If you

Rule 3.6 Place a non-breaking space (~) right before references.

This is actually a slight simplification of the real rule, which should invoke common sense. Place non-breaking spaces where a line break would look 'funny' because it occurs right in the middle of a construction, especially between a reference type (Chapter) and its number.

3.3 Choice of 'fonts'

Professional typography distinguishes many font attributes, such as family, size, shape, and weight. The choice for sectional divisions and layout elements has been made, but you will still occasionally want to switch to something else to get the reader's attention. The most important rule is very simple.

Rule 3.7 When emphasising a short bit of text, use `\emph`.

In particular, *never* use bold text (`\textbf`). Italics (or Roman type if used within italics) avoids distracting the eye with the huge blobs of ink in the middle of the text that bold text so quickly introduces.

Occasionally you will need more notation, for example, a consistent typeface used to identify algorithms.

Rule 3.8 Vary one attribute at a time.

For example, for WeirdSort we only changed the shape to small caps. Changing two attributes, say, to bold small caps would be excessive (LaTeX does not even have this particular variation). The same holds for mathematical notation: the reader can easily distinguish g_n, $G(x)$, $\mathcal{G}$ and G.

Rule 3.9 Never underline or uppercase.

No exceptions to this one, unless you are writing your thesis on a typewriter. Manually. Uphill both ways. In a blizzard.

3.4 Displayed equations

Rule 3.10 Insert paragraph breaks *after* displays only where they belong. Never insert paragraph breaks *before* displays.

LaTeX translates sequences of more than one linebreak (i.e., what looks like an empty line in the source code) into a paragraph break in almost all contexts. This also happens before and after displays, where extra spacing is inserted to give a visual indication of the structure. Adding a blank line in these places may look nice in the sources, but compare the resulting display

$$a = b$$

to the following:

$$a = b$$

The first display is surrounded by blank lines, but the second is not. It is bad style to start a paragraph with a display (you should always tell the reader what the display means first), so the rule follows.

Rule 3.11 Never use `eqnarray`.

It is at the root of most ill-spaced multiline displays. The *amsmath* package provides better alternatives, such as the `align` family

$$\begin{aligned} f(x) &= \sin x, \\ g(x) &= \cos x, \end{aligned}$$

and `multline` which copes with excessively long equations:

$$\begin{multline} \mathrm{P}\big[X_{t_0} \in (z_0, z_0 + dz_0], \ldots, X_{t_n} \in (z_n, z_n + dz_n]\big] \\ = \nu(dz_0) K_{t_1}(z_0, dz_1) K_{t_2 - t_1}(z_1, dz_2) \cdots K_{t_n - t_{n-1}}(z_{n-1}, dz_n). \end{multline}$$

3.5 Floats

By default this style provides floating environments for tables and figures. The general structure should be as follows:

```
\begin{figure}
  \centering
  % content goes here
  \caption{A short caption}
  \label{some-short-label}
\end{figure}
```

Note that the label must follow the caption, otherwise the label will refer to the surrounding section instead. Also note that figures should be captioned at the bottom, and tables at the top.

The whole point of floats is that they, well, *float* to a place where they fit without interrupting the text body. This is a frequent source of confusion and changes; please leave it as is.

Rule 3.12 Do not restrict float movement to only 'here' (`h`).

If you are still tempted, you should avoid the float altogether and just show the figure or table inline, similar to a displayed equation.

Chapter 4

Generative Graph Models (GGM)

4.1 Generative Graph Models (GGM)

[Bläsius et al., 2018] compares various generative graph models: Erdos-Renyi (ER), Barabasi-Albert (BA) preferential attachement graphs, Chung-Lu and hyperbolic graphs. They are compared on the basis of their ability to simulate real graphs of interest, such as social networks, citation networks, and biological networks, which are known to have power law node degree distributions.

4.1.1 GGM 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 $V = [n]$ and each $u \sim v$ iid with probability p.

4.1.2 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 likewise $\hat{p} = |E|/\binom{n}{2}$.

Fitting $\hat{\theta}$ could be done by likelihood estimation. This is tractable for the Erdos-Renyi GGM for instance, by solving $\arg\min_p \sum_{u \neq v} \log(p^{e_{uv}}) + \log((1-p)^{1-e_{uv}}))$. For GIRGs however, this would be intractable. For example, trying to calculate $p(G|\alpha, \boldsymbol{w}, d) = \int_{\boldsymbol{x} \in [0,1]^d} p(G|\alpha, \boldsymbol{w}, \boldsymbol{x}) p(\boldsymbol{x})$ is not feasible.

Instead we can use a heuristic method based on Approximate Bayesian Computation. We seek the parameter $\hat{\theta}$ that minimises the expected distance $\mathbb{E}[d(G, G' \sim \mathscr{G}(\theta))]$ for some distance metric d. For d to be effective, ideally it would be something like $(f(G) - f(G'))^2$, perhaps ideally for f a sufficient statistic of θ.

4.1.3 Plausibility of Fitted GGMs

Similarly to the use of a statistic f as a proxy for fitting $\hat{\theta}$, we can use other statistics f' to evaluate the plausibility of a fitted GGM. Which statistic(s) f should be used for ABC fitting, and which for plausibility analysis is unclear to me.

Once a GGM is fit, we can then produce more graphs from it, via $G' \sim \mathcal{G}(\hat{\theta})$. Clearly in general G and G' will be quite different due to the inherent randomness of the GGM. If G really had been produced by $\mathcal{G}(\hat{\theta})$ though, we might expect G and G' to have relatively similar statistics, e.g. $\mathbb{E}[|E'|] = \hat{p}\binom{n}{2}$ with some small variance.

Furthermore if we were to generate a whole list of graphs $G_1, \ldots, G_k$ from $\mathcal{G}(\hat{\theta})$, we would expect G not to in any way stand out from the crowd.

In fact, in the best case, actually $G \sim \mathcal{G}(\theta^*)$, so our parameter estimate is $\hat{\theta} = \theta^* + \varepsilon$. Here ε is a random noise in our ability to fit θ, hopefully small and with mean zero.

Then for any statistic f, we could consider the randomness $f(G) \sim F_1$, and $f(G') \sim F_2$. Due to the ε error in θ estimation, $F_2|\hat{\theta}(G) \nsim F_1$, but we would hope that $f(G)$ would still be reasonably within distribution, and less strongly (???) that $F_2 \sim\approxeq F_1$ not too far off, rather likely a higher variance but otherwise similar distribution.

Hence plausibly sampling $G_1, ..., G_k \sim \mathcal{G}(\theta^*)$, fitting $\hat{\theta}_i(G_i)$ and further sampling $G'_i \sim \mathcal{G}(\theta^*)$, we expect that $f_i \sim F_1$ would be hard to separate from $f'_i \sim F_2$ with a classifier.

4.1.4 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 $G_1, ..., G_k$ are generated from a particular GGM $\mathcal{G}$, but with differing parameters: $G_1 \sim \mathcal{G}(\theta_1^*), ..., G_k \sim \mathcal{G}(\theta_k^*)$.

They again fit individual $\mathcal{G}(\hat{\theta}_i)$ with which to generate one G'_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 again done by comparing a wide number of statistics/metrics compted for each graph. These "features" include measures such as the number of nodes, average node degree, centrality, closeness, diameter, clustering coefficient and so on.

(QUESTION: why not make multiple G'_{ij} per real graph G_i? I.e. just doing the previous section for each real Graph? Presumably to reduce computation?)

This gives us a mirrored real/fake graph features dataset: $\boldsymbol{f}_i = f(G_i)$ for feature vector function $\boldsymbol{f}$, against $\boldsymbol{f}'_i$. A classifier is trained on this dataset, to classify a given graph as real/fake. This classifier 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 $\tilde{\mathscr{G}}$.

Contextually, GIRGs are compared with other generative graph models: Erdos-Renyi (ER), Barabasi-Albert (BA) preferential attachement graphs, Chung-Lu and hyperbolic graphs. In [Bläsius et al., 2022]

Chapter 5

GIRG generation

5.1 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

TODO below (div by n, infty norm) is what I had before, but in fact bringmann2019geometric uses below with div by W and ambiguous norm.

$$p_{uv} = \Theta\left(\min\left\{1, c\left(\frac{w_u w_v}{n\left\|x_u - x_v\right\|_\infty^d}\right)^\alpha\right\}\right) \tag{5.1}$$

$$p_{uv} = \Theta\left(\min\left\{1, c\left(\frac{w_u w_v / W}{\left\|x_u - x_v\right\|^d}\right)^\alpha\right\}\right) \tag{5.2}$$

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 or cube of side length 1.

This edge probability has a geometric factor from the distance $r_{uv} = \|x_u - x_v\|_\infty$, and is proportional to $r_{uv}^{-d\alpha}$, where $\alpha > 0$, meaning that close by nodes are more likely to be connected. The n^{-1} normalising factor ensures that as $n \to \infty$, expected node degrees are unchanged.

We often write $\rho_{uv} = \left(\frac{w_u w_v / W}{\|x_u - x_v\|^d}\right)$ to save on writing.

power law degree sequence Like Chung-Lu, the weight sequences are usually assumed to follow a powerlaw distribution with an exponent $\tau \in$

$(2,3)$ (at least within some tolerance and in the large weight tail).

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

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

Similarity to Chung-Lu A key property of the GIRG model is that marginalising over possible node locations gives that $E_{x_v}[p_{uv}|x_u, w_u, w_v] = \Theta(\min\{1, \frac{w_u w_v}{n}\})$, which matches the Chung-Lu GGM. This means that many results for Chung-Lu carry through automatically into GIRGs, like simple facts that $E[d_u] = \Theta(w_u)$ (the expected degree of a node of a fixed weight is proportional to that weight). For this to hold we actually need to have $\alpha > 1$ - otherwise there are too many long distance edges. Essentially $\alpha > 1$ depresses the number of long distance edges to be $\Theta(\min\{1, \frac{w_u w_v}{W}\})$. No matter how large α is, there will always be $\Theta(\min\{1, \frac{w_u w_v}{W}\})$ short distance edges: $u \sim v$ where $p_{uv} = 1$.

Geometry The exact geometry χ, and the distance function $||\cdot||$ can vary a great deal without affecting the key properties of GIRGs. $\chi = \mathbb{T}_d$ the d-dimensional torus is very handy for proofs, as the viewpoint of any node x_u is equivalently at the "centre" of the space. For real applications the d-dimensional cube $\chi = [0,1]^d$ can be more realistic, but then if x_u is at the edge of the cube it has a different viewpoint to a node more at the centre of the cube.

Taking $||\cdot|| = ||\cdot||_\infty$ is also useful, but can be replaced equivalently by euclidean or other norms. The minimum component distance $||x|| = \min_i |x_i|$ is not a norm, but still holds retains most of the GIRG properties. This can make sense in that two nodes might have a higher edge probability by being close in one dimension, rather than every dimension.

The reason for the power of d in r_{uv}^d is to keep the edge probabilties equivalent with respect to the volume of space in different dimensions. The more general formula replaces replaces $||x_u - x_v||^d = r_{uv}^d$ with $Vol(B_r)$ the volume of the ball of radius $r = r_{uv}$. For norms, $Vol(B_r) = \Theta(r^d)$, e.g. in the ∞-norm, $Vol(B_r) = (2r)^d$ as a cube with side-length $2r$. The euclidean ball has some π's in the formula. For the minimum component distance, $Vol(B_r) = \Theta(r)$. Volumetric equivalence makes sense in that the edge probability $p_{uv}|x_u, w_u, w_v$ in the Torus is determined by integrating $\int_{r=0}^{r=1} p_{uv}(r)p(r)dr =$

$\int_{Vol=0}^{Vol=1} p_{uv}(Vol)dVol$, where $Vol(r) = Vol(B_r)$ is the volume of the ball of radius r. Hence across different GIRGs of different dimensions d, keeping $p_{uv}(Vol)$ the same function therefore keeps the pairwise edge probabilities p_{uv} the same (but not the joint edge probability distribution of course).

5.2 C++ GIRGs

We use the C++ implementation of GIRG generation by [Bläsius et al., 2022]. They implement the algorithm in [Bringmann et al., 2019], which they claim to have linear runtime. A GIRG, having power law weighted nodes, is shown to have $\Theta(1)$ average degree. Hence there are $\Theta(n)$ edges, so presumably this means that the runtime is not much different from an oracle writing out the edges sequentially.

We use this code so much so will need to take note of their preferred notation:

$$p_{uv} = \min\left\{1, c\left(\frac{w_u w_v / W}{\|x_u - x_v\|_\infty^d}\right)^\alpha\right\} \tag{5.3}$$

Where they take $x_u \in \chi = \mathbb{T}_d$ the torus. They use a variant of the GIRG where normalisation by n is replaced by that of $W = \sum_{u \in V} w_u$. For w_u obeying a $\tau > 2$ power law distribution, $W = \Theta(n)$ with high probability, so this is fine.

The inner constant c in the C++ implementation still falls into the wider GIRG definition of $p_{uv} = \Theta\left[\min\{...\}\right]$, as p_{uv} always lies in the interval $c\min\left\{1, \left(\frac{w_u w_v/W}{\|x_u - x_v\|_\infty^d}\right)^\alpha\right\} \leq p_{uv} \leq \min\left\{1, \left(\frac{w_u w_v/W}{\|x_u - x_v\|_\infty^d}\right)^\alpha\right\}$, for $c \leq 1$, and with the upper and lower bounds swapped for $c > 1$. I.e. the wider GIRG definition just requires that there are some lower and upper bounding constants c_L, c_U such that for every pair of nodes u, v, their edge probabilities are given as $c_L \min\{1, (...)^\alpha\} \leq p_{uv} \leq c_U \min\{1, (...)^\alpha\}$. If these bounds are fixed for increasing n, then all the nice properties of GIRGs can be proven!

5.2.1 Fitting GIRGs for Blasius evaluation framework

A GIRG parametric model is GIRG(n, d, c, α, τ). We fit the model to a certain real graph instance $G = (V, E)$ in a few steps. Our method used is a form of Approximate Bayesian Computation (ABC). In short, ABC fits a parametric bayesian model to data D without computing a posterior likelihood $p(\theta|D)$ (infeasible), rather by sampling θ from the prior, and accepting θ if the simulated data D' from θ is "close enough" to D, under some distance metric $\rho(D, D')$.

In our case we fit θ given just one datapoint $D = G$, and we use different distance metrics ρ for fitting each part of $\theta = (n, d, c, \alpha, \tau)$

Fitting number of nodes n

We actually follow [Bläsius et al., 2018] which first preprocesses $G \leftarrow \text{shrinkToGCC}(G)$ before fitting $\mathcal{G}$ to it, by shrinking G to its largest connected component. Then we fit $n = |V|$ straightforwardly.

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 they share the property of whp producing a unique giant component (TODO check is this true) - one with a linear number of nodes. Hence if any real graph had multiple giant components (say e.g. $V = A \sqcup B \sqcup (...)$ with $|A| = n/2$, $|B| = n/3$), then we would not expect any of our GGMs to fit well to the whole graph.

Restricting to a connected component has the added benefit of making some graph statistics more meaningful/sensical - for instance diameter and path lengths. This explains why Blasius even further post-processes the GGM generated fake graphs to also restrict to their largest connected component. Blasius goes as far as to, for the hyperbolic GGM, using a fitting algorithm that actually estimates a higher number of nodes $n > |V|$ in order to approximately have the largest connected component of $G' \sim \mathcal{G}$ be of size $|V|$. We don't do this for our GIRGs however, as we find this algorithm prone to error, and unnecessary, at least for the socfb Facebook graphs.

As an example here are four facebook graphs:

Note that GIRGs might still do the best job of fitting multiple large components however, as they have the property of containing sublinear separators - essentially if you divide the torus with a hyperplane (or divide out a subcube) to produce $V = A \sqcup B$, then whp A has sublinear of $|A|$ edges to B (despite having linear number of edges to itself). And indeed the geometrically restricted A subgraph is stochastically still a (smaller) GIRG, just one with a non-toroidal geometry.

Fitting power law exponent τ for weight sampling

We fit τ to the tail of the degree distrbition of G, using the python package `powerlaw`.

The degree distribution of the graph G is given 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 with power law exponent 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 $dd(x) \propto x^{-\tau}$.

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 5.1: 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).

A brief sketch of why this occurs:

We saw previously that regardless of the GIRG geometry, $P(u \sim v|w_u, w_v) = \Theta(\frac{w_u w_v}{n})$.

More specifically, this gives that $P(u \sim v|w_u) = p_u = \Theta(\frac{w_u}{n})$, such that the degree $d_u \sim Bin(n-1, p_u)$. WHP, as $n \to \infty$, $\forall u,\ w_u = o(n)$ s.t. $p_u = o(1)$ and so d_u converge in distribution to $Poisson(\Theta(w_u))$. This means that $\mathbb{E}[d_u] = \Theta(w_u)$, and so $\mathbb{E}[dd(x)] = \mathbb{E}[\frac{\sum_u 1_{d_u=x}}{n}] = \mathbb{E}[d_u = x]$. This we can calculate as $\int_w P(w_u = w)P(d_u = x|w_u = w)dw = \int_w w^{-\tau}\mathbb{P}(\text{Poisson}(\Theta(w)) = x)dw$. It's essentially the inner product (integral of product) of two probability distributions, the first is $w^{-\tau}$, the second is a Gaussian looking like function with mean and variance $\Theta(w)$, and hence to first order the integral is $\Theta(w^{-\tau})$ (e.g. if you changed the gaussian integral to have the same mean w but variance squeezed $\sigma^2 \to 1$ as to be like a delta function). It follows that for large x, $\mathbb{E}[dd(x)] = \Theta(x^{-\tau})$.

Therefore for large degrees x, we can fit the GIRG power law exponent to the tail of the degree distribution of G. `powerlaw` does this by first finding a lower bound x_{min} for the power law behaviour, and then fitting τ to the tail $x > x_{\text{min}}$.

For a given x_{min}, τ is fit on the resultant degree distribution tail using maximum likelihood estimation. The optimal x_{min} is chosen to minimise the Kolmogorov-Smirnov distance between the power law fit and the resultant tail's emprirical degree distribution.

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

Power Law Distribution

A power law distribution $x \sim \text{powerlaw}(\tau)$ simply has pdf $p(x) \propto x^{-\tau}$ with support $x \in [1, \infty]$, i.e. default $x_{\text{min}} = 1$.

Power Law alternative: weight copying

Generating weights $w_u \overset{iid}{\sim} \text{powerlaw}(\tau)$ is fine as a model prior, however it's not a good fit to real world data. A sequence of weights with a constant q-tile of largest weights fitting into τ powerlaw tramlines fits the general GIRG formulation, but could still look quite different based on the distribution of the smaller weights. An easy improvement for fitting a specific real graph is to take the sequence of node degrees as weights (these are now all positive integers ≥ 1 as opposed to real numbers $\geq x_min > 0$, since in the GCC minimum degree is 1). This would clearly have better classificaiton performance than power law generating weights.

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). Weight copied GIRGs become a "fair" comparison against weight copied chung-lu: the only difference being random point locations in some geometric space. It is less fair in comparison to the other ER and BA GGM's which fit on a much courser metric of average degree.

Hence we tried both power law generated and directly copied weights for our GIRG fitting, so as to be able to more fairly compare against the different GGMs. Comparing weight copied chung-lu with weight copied GIRGs was additionally interesting as their classification from real graphs was harder and hence more informative (trying to compare a 97% vs 99% classification accuracy is less meaningful than a 80% vs 90%).

(QUESTION: does the GIRG formulation actually guarantee that as n -¿ infinity you couldn't tell the difference? I don't think so. E.g. if 1/4 of nodes had w=1, 1/4 had w=10 and 1/2 were power law above 10 distributed.)

Fitting c and α

These are our last two parameters to fit. The parameter c makes sense as a proxy for the average degree, whereas the parameter α is more linked to the power of the geometry - larger α decreases the edge probabilities of edges long enough to have $\rho_{uv} = \frac{w_u w_v}{nVol(r_{uv})} < 1$, with sharper effect for $\rho_{uv} \ll 1$.

Therefore we fit c for a given α so as to match the average degree of G. We then fit α for a given c to match the local clustering coefficient (LCC) of G which is similarly linked to the power of the geometry - a sharper distance cutoff increases the LCC. Fitting both simultaneously looks like coordinate ascent in 2D, we alternatively maximise $c \leftarrow \hat{c}_1;\ \alpha \leftarrow \hat{\alpha}_1;\ c \leftarrow \hat{c}_2;\ \alpha \leftarrow \hat{\alpha}_2;\ ....$

[Bläsius et al., 2018] gives a method to fit c given α, d, and a pre-sampled set of weights $(w_u)_{u \in V}$. They derive a formula for the expected average degree $\mathbb{E}[\overline{deg}]$ of the GIRG, which essentially looks like:

$$f(c) = c \cdot A(\cdots) + c^{1/\alpha} \cdot B(\cdots) \tag{5.4}$$

They then numerically solve for eq. (5.4) by $\hat{c} : f(\hat{c}) = \overline{deg}$ for desired average degree $\overline{deg}$. This method miraculously works for all Volume based GIRGs (regardless of exact disance function $r(x_u, x_v) = r_{uv}$), and we can adapt their formula to not care about dimension d either.

Derivation:

Recall the formula,

$$p_{uv}(r) = \min\left\{1, c\left(\frac{w_u w_v}{nVol(r_{uv})}\right)^{\alpha}\right\}; \quad \rho_{uv} = \frac{w_u w_v}{nVol(r_{uv})} \tag{5.5}$$

The whole point of the above formulation is to make such that $p(u \sim v | x_u, w_u, w_v) = \Theta(w_u w_v / n)$, regardless of geometry. We'll derive the same resultant $p(u \sim v | w_u, w_v) = \mathbb{E}_r[p_{uv}(r)]$ regardless, relying only on the fact that $Vol(r)$ is an increasing function. Finally, $\mathbb{E}[d_u] = \sum_v p(u \sim v | w_u, w_v)$ and hence $\mathbb{E}[\overline{deg}] = \sum_u \mathbb{E}[d_u]$.

Now $p(u \sim v | x_u, w_u, w_v) = \int_r p_{uv}(r) p(r) dr$. With our volume function $Vol(r)$, we break this down into $\int_{r:\rho_{uv} \geq 1} + \int_{r:\rho_{uv} < 1}$. We write $\hat{r} : Vol(\hat{r}) = \frac{w_u w_v}{n} c^{1/\alpha}$.

Hence we get $\int_0^{\hat{r}} p(r) dr + \int_{\hat{r}}^{r_{\max}} p_{uv}(r) p(r) dr$.

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

$$\int_0^{Vol(\hat{r})} dVol + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} p_{uv}(Vol)dVol \tag{5.6}$$

$$=Vol(\hat{r}) + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} c\left(\frac{w_u w_v}{n}\right)^{\alpha} Vol^{-\alpha}dVol \tag{5.7}$$

$$=Vol(\hat{r}) + c\left(\frac{w_u w_v}{n}\right)^{\alpha} \frac{1}{\alpha - 1}\left[\left(\frac{w_u w_v}{n}\right)^{1-\alpha} c^{\frac{1-\alpha}{\alpha}} - 1\right] \tag{5.8}$$

$$=c^{1/\alpha}\left(\frac{w_u w_v}{n}\right)\left[1 + \frac{1}{\alpha - 1}\right] - \frac{c}{\alpha - 1}\left(\frac{w_u w_v}{n}\right)^{\alpha} \tag{5.9}$$

Where in the last two lines we sub in $Vol(\hat{r}) = c^{1/\alpha}\left(\frac{w_u w_v}{n}\right)$, and $Vol(\mathbb{T}) = 1$. Following the Blasius Appendix, the formula just needs a correction for pairs u, v such that $Vol(\hat{r}) = c^{1/\alpha}\left(\frac{w_u w_v}{n}\right) > 1$, wherein the second integral is unnecessary, and the first is capped lower at 1, the volume of the Torus - i.e. for such u, v, their weights are large enough so that $p_{uv}(r) = 1$ identically, no matter where x_u, x_v are placed in the Torus.

Finally the average degree formula in eq. (5.4) is just averaging over all terms in eq. (5.9) for each pair of vertices u, v.

5.3 Cube GIRGs Section

Cube GIRGs are an alternative formulation to the normal Toroidal GIRGs. Instead of the d-dimensional torus $\mathbb{T}$, we just use the d-dimensional cube $[0, 1]^d$. Thus we lose the symmetry and simplicity of the torus.The hopeful benefit is that this generate more realistic graphs, as few situations in the real world are Toroidal.

For example with social network graphs, likely geometric quantities are home address (as a 2D point, since these networks are generally from a small town/city/university - only on planet earth scale do locations become somewhat more toroidal), political leaning (typically a 2D axis of economic left/right, and authoritarian/libertarian - e.g. an extreme authoritarian is unlikely to get along with an extreme libertarian), athletic proclivity etc. all generally are non toroidal. Perhaps one toroidal-esque quantity could be that in all cases, a pair of positive and negataive far out leaning people share in common their marginalisation / non conventionality - this could potentialily be captured on a 1D non toroidal normie vs hipster scale.

5.3.1 Cube GIRG Formulation

The neat geometric symmetry of volume and distance breaks down when translating to Cube GIRGs. The distance r^C_{uv} vs $r^{\mathbb{T}}_{uv}$ in a cube vs torus satisfies $r^C_{uv} \geq r^{\mathbb{T}}_{uv}$. For the ∞ norm, $r^C_{uv} = r^{\mathbb{T}}_{uv} \iff r^C_{uv} \leq \frac{1}{2}$. The simplest edge probability translation therefore is just to use the same $r_{uv}^{-d\alpha}$ factor, such that Cube GIRGs are just a more sparse version of Toroidal GIRGs. We call this the distance based formulation.

An alternative volume based formulation, seeking to replicate the $r_{uv}^d \propto Vol(B_{r_{uv}})$, is complicated by the lack of symmetry. The whole point of the general volume formulation of edge probability was that the node u has a set edge probability to neighbours v at a certain volumetric distance - where now in the cube case, we could take $Vol(B_{r_{uv}}(u)) = Vol(B_{r_{uv}}(u)) \cap [0,1]^d$, i.e. volume only counts within the cube itself. This is not symmetric in that $Vol(B_{r_{uv}}(u)) \neq Vol(B_{r_{uv}}(v))$: if u is closer to the edge of the cube and v closer to the centre, then $Vol_u < Vol_v$. However dealing with undirected edges, we must have one agreed upon formula for $p_{uv}(r)$, and so a natural formula would be to replace $Vol(B_{r_{uv}}) \mapsto \sqrt{Vol(B_{r_{uv}}(u))Vol(V_{r_{uv}}(v))}$. This would basically seek to compensate a u near the edge, allowing it to better seek far away neighbours. Even still, this does not end up with all nodes of a weight having the same expected number of neighbours regardless of their location. Rather it's like we had directed eges, yielding undirected edges if both directions are present: $u \rightarrow v$ and $v \rightarrow u$ implies $u \sim v$. With this volume based formulation, every node has out edges based on its own weight and volumetric distance to other nodes. $p(u \sim v) = p(u \rightarrow v)p(v \rightarrow u)$. If we ignored the minimum term in the edge probabilities (i.e. saying that $p(u \rightarrow v) \propto w_u / \sqrt{Vol(B_{r_{uv}}(u))}$, not upper bounded by 1, then actually all nodes would have the same expected number of neighbours, regardless of their location.

In the social network analogy, the distance based formulation is like saying that people make friends only with those who live close by, and so if you're a person who lives far off in a remote mountain village, you're unlikely to have many friends. The volume based formulation is saying that all people, no matter how extreme their geometric location, have the same desire to make friends. So even if you're the most far left communist, you will still make as much total social effort, primarily on far lefters, but also towards even up to centre left potential friends. However to most central leftists, the communist is too extreme and won't get much social budget compared to other more similar folks.

TODO think more if good idea? Another possible model for spheres of influence about nodes near the edge of the cube is one where spheres of influence are no longer radially symmetric. In the volume based formulation in a 2D space, where u is near the north edge of the square, efforts to find friends in the north are redirected equally along the east, south and west at the same radius. In a physical analogy, it's possible instead that "feelers" to the north are redirected along the path of least resistance, to the east and west. In a political spectrum analogy, more aligned to the min GIRG philosophy, if a person is located in the far social left of the cube, but economically central, they may seek friendships amongst those who are either similarly economically or similarly socially. Hence if there are no people more socially left, they may redirect their social dimension friendship effort budget to those a bit more socially right of themselves, rather than also to those economically left/right close by. We don't pursue this model!

5.3.2 Cube GIRG generation - coupling

We elect to go for the distance based cube GIRG formulation - partly because it permits a simple coupling based generation algorithm, which runs in $O(n)$ extra time on top of the initial Torus GIRG generation.

The algorithm is simple:

Algorithm 1 Generate Cube GIRG from Torus via coupling

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

5.3.3 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 n, d, τ, 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 $G_i \sim \mathrm{GIRG}(c_i)$ and setting $c_{i+1} \leftarrow c_i \frac{n\overline{deg}}{2|E(G_i)|}$ until convergence.

5.4 Blasius classification results

TODO

1. table of classification accuracies vs real-world

2. explanation of copyweight GIRGs, normal GIRGs, cube GIRGs, min GIRGs, mixed GIRGs
3. analysis of which features different models seem to be good at
4. table of classification accuracies vs 1D GIRGs
5. analysis of which features different dimensional GIRGs seem to be distinguishable on
6.

Chapter 6

Diffusion Maps

6.1 Introduction

Diffusion Maps are a technique for discovering underlying geometricity of a graph $G = (V, E)$ solely from the connectivity. We will put the technique to use in order to invert the GIRG generative process - go from a graph produced by a d-dimensional GIRG, and infer the original locations $(x_u)_{u \in V}$ of the vertices $x_u \in \mathbb{T}^d$.

The idea of Diffusion Maps is to analyse the diffusion process of random 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

$$M_{ij} := P(X(t+1) = j | X(t) = i) = \frac{w_{ij}}{\deg(i)} \tag{6.1}$$

$$\begin{aligned}
M &= D^{-1}W && \text{transition matrix}\\
D_{uu} &= \sum_{v\in V} W_{uv} && \text{diagonal degree matrix}\\
W_{uv} &= \begin{cases} 1 & u \sim v \\ 0 & u \nsim v \end{cases} && \text{adjacency matrix}\\
S &= D^{-1/2}WD^{-1/2} && \text{symmetric matrix}\\
&= V\Lambda V^T && \text{diagonalisation into orthonormal e-vectors}\\
\Phi &= D^{-1/2}V = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, ..., \boldsymbol{\phi}_n] && \Psi = D^{1/2}V = [\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, ..., \boldsymbol{\psi}_n]\\
\Phi^T\Psi &= \Psi^T\Phi = I_{n\times n} && \text{due to orthonormality of } V
\end{aligned}$$

We use the diagonalisation of S to write M as

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

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 $\boldsymbol{\phi}_1 = c\mathbf{1}$.

Then the diffusion map representation of a node is then

$$\begin{aligned}
e_i^T M &= \sum_{k=1}^{n} \phi_i(k)\lambda_k^t \boldsymbol{\psi}_k && \text{diff map of node } i \text{ after } t \text{ steps}\\
i &\mapsto (\phi_2(i)\lambda_2^t, ...\phi_{m+1}(i)\lambda_{m+1}^t) && \text{m-truncated diff map representation}
\end{aligned}$$

The truncated diffusion map summarises the distinguishing features of a node's random walk cloud after t steps, by taking the scale factor of $\boldsymbol{\psi}_k$ ($\boldsymbol{\psi}_k$ itself is ignored as it's relatively normalisesd: $\boldsymbol{\psi}_k = \boldsymbol{d}^{1/2} \odot \boldsymbol{v}_k$ (TODO is this really normalised?)). 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 $\boldsymbol{\pi}_i = \frac{d_i}{\sum_j d_j}$.

We attempt to improve the diffusion map process for GIRGs. In particular, we'd like the random walk to prioritise edges going to nodes with low degree, as these are more likely to be the neighbours of the node in the underlying geometry. We do this by modifying the transition matrix M to $\tilde{M}$ as follows:

$$\tilde{M} = \tilde{D} M D^{-\gamma}$$

Where $\tilde{D}$ is to normalise $\tilde{M}$ to ensure that it's still a row stochastic matrix: $\sum_j \tilde{M}_{ij} = 1$. The tuning parameter $\gamma > 0$ directly controls the bias towards low degree nodes: $M_{ij} d_j^{-\gamma}$ means that nodes j with high d_j have a relatively lowered transition probability. This unfortunately messes up all our nice equations. $\tilde{M}$ is still row stochastic, which means it still has a largest magnitude eigenvalue $\lambda_1 = 1$, and by the Gershgorin circle theorem, all other $|\lambda_i| \leq 1$.

We can write it as $\tilde{M} = D_1 \Phi \Lambda \Psi^T D^{-\gamma}$. This does at least allow the decomposition $\tilde{M} = \sum_k \lambda_k \tilde{\boldsymbol{\phi}}_k \tilde{\boldsymbol{\psi}}_k$, where $\tilde{\boldsymbol{\phi}}_k = D_1 \boldsymbol{\phi}_k$ and $\tilde{\boldsymbol{\psi}}_k = D^{-\gamma} \boldsymbol{\psi}_k$. In particular, $\tilde{M}(D_2^{-1} \boldsymbol{\phi}_k) = \lambda_k D_1 \boldsymbol{\phi}_k$. This does not mean that $\tilde{M}$ has eigenvalues λ_i, rather they're slightly different.

Instead we hope that $\tilde{M}$ is diagonalisable as $\tilde{M} = BAB^{-1}$ for diagonal A, and use the corresponding diffusion map representation $i \mapsto (\boldsymbol{b}_2(i) a_2^t, ... \boldsymbol{b}_{m+1}(i) a_{m+1}^t)$.

6.2 Diffusion Maps on GIRGs

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 estimate for the original geometric location of the node.

If d is unknown, diffusion maps present one way to infer the dimensionality by analysing the ordered sequence of eigenvalues $\lambda_2 < \lambda_3 < ....$ In theory there is a good cutoff point whereby the first d eigenvalues are of similar large size, and the rest are much smaller. This indeed works well for graphs synthetically generated from GIRGs, not so well on real world graphs.

We see in fig. 6.2b that the inferred first two diffusion map coordinates do fit a square quite well. 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 $\boldsymbol{\phi}_1 = k\mathbf{1}$: 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.

6.2.1 Rescaling/Shifting Diffusion Maps

The 0 centering can easily be fixed to be more Cube GIRG like by shifting the points $(\boldsymbol{x}_u)_i \leftarrow (\boldsymbol{x}_u)_i + \min_v (\boldsymbol{x}_v)_i$. If we are certain that the points should be distributed within the unit cube, then we can simply rescale separately along each dimension: $x \leftarrow \frac{x - x_{\min}}{x_{\max} - x_{\min}}$. If furthermore we're certain that the distribution within the unit cube should be relatively uniform, we can perform a coordinatewise "uniformify" procedure that replaces $(x_u)_i$ with its percentile value compared with other $(x_v)_i$. fig. 6.3 shows the "uniformify" procedure in action. Notably on real graphs where the truncated diff map coordinates are not guaranteed to be independently distributed, coordinatewise percentile mapping can lead to slightly odd results - see socfb-Amherst where there's a strong $x_1 = 1 - x_2$ correlation for small x_1. It's still a good improvement over the original non rescaled diffusion map.

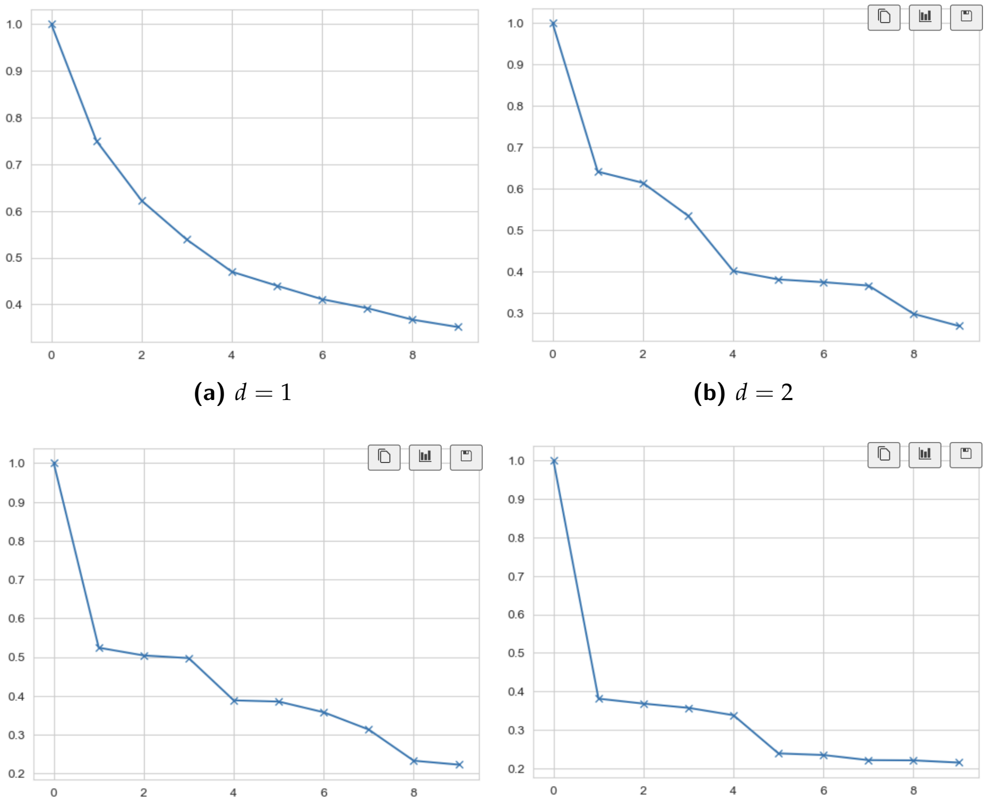
Critically since there is no guarantee that the scaling of diffusion map coordinates is the same as the original GIRG coordinates, using some kind of prior knowledge to rescale the diffusion map is important to yield geometric information with meaningful inter-point distances.

Rotated Points One issue with diffusion maps as seen in fig. 6.2b, 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 and uniformifying to the unit cube a little questionable.

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

In this light 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. The relative flatness of $\lambda_2, \lambda_3, \lambda_4, \lambda_5$ line segment in fig. 6.1d is a testament to the underlying cube 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.



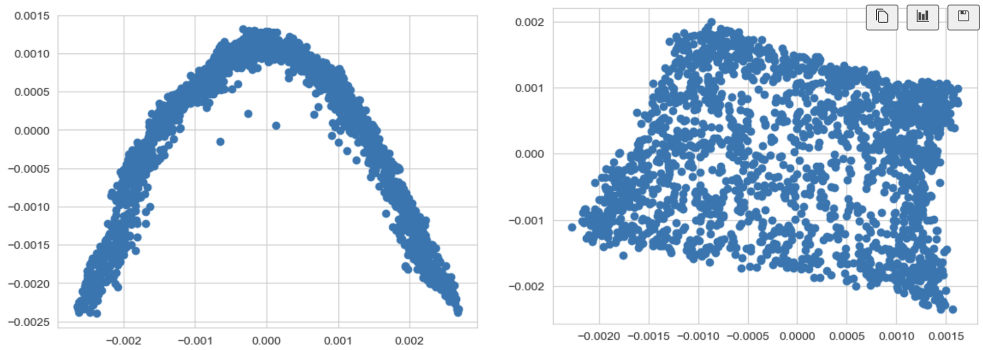
(a) $d = 1$

(b) $d = 2$

(c) $d = 3$

(d) $d = 4$

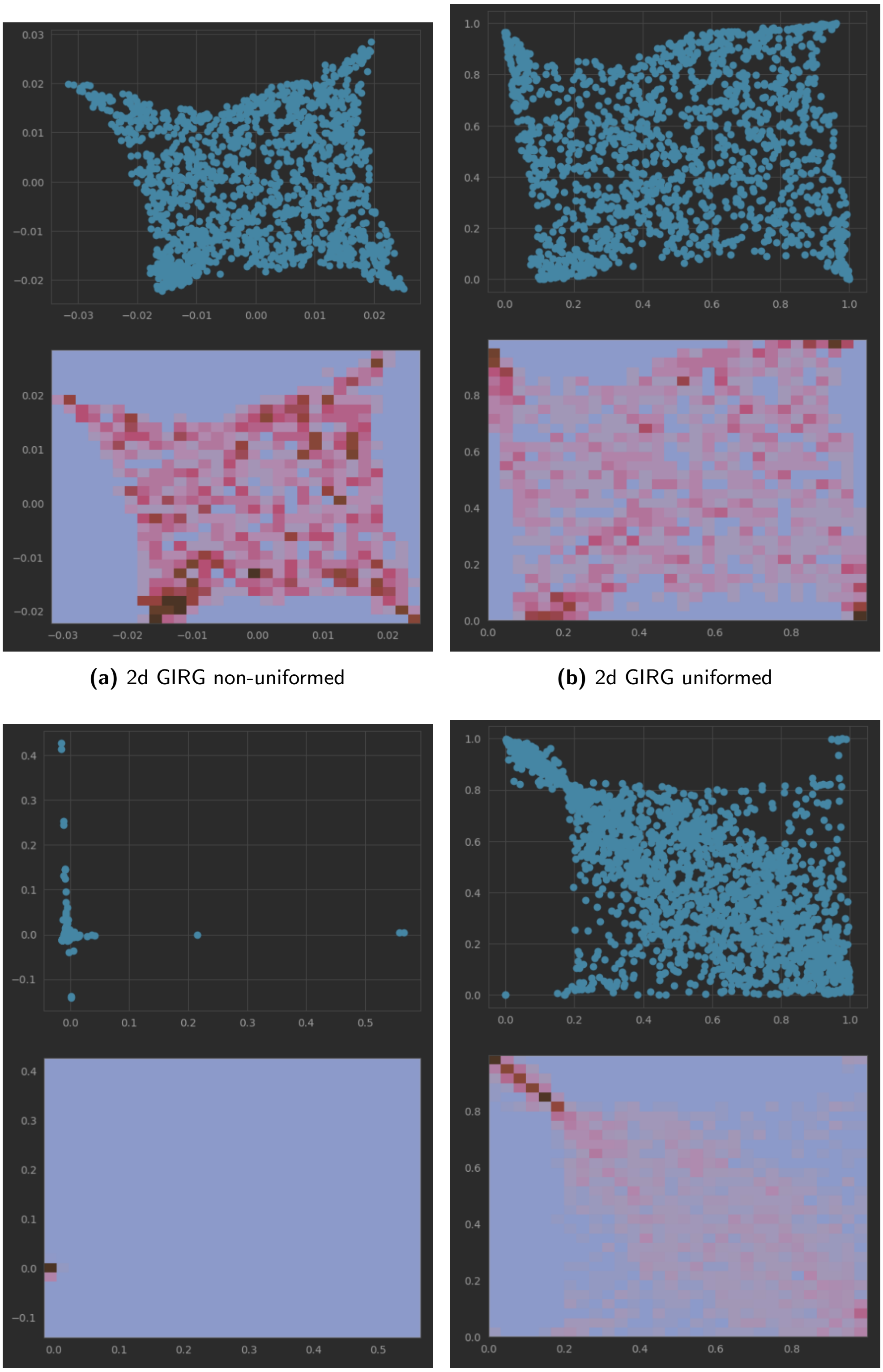
Figure 6.1: Diffusion map eigenvalues (including $\lambda_1 = 1$) for a $n = 2000, \tau = 2.5, \alpha = 1.3$ Cube GIRG with $d = 1, 2, 3, 4$ dimensions.



(a) $d = 1$

(b) $d = 2$

Figure 6.2: Diffusion map scatter plot of the first two extracted coordinates from 1d and 2d GIRGs.



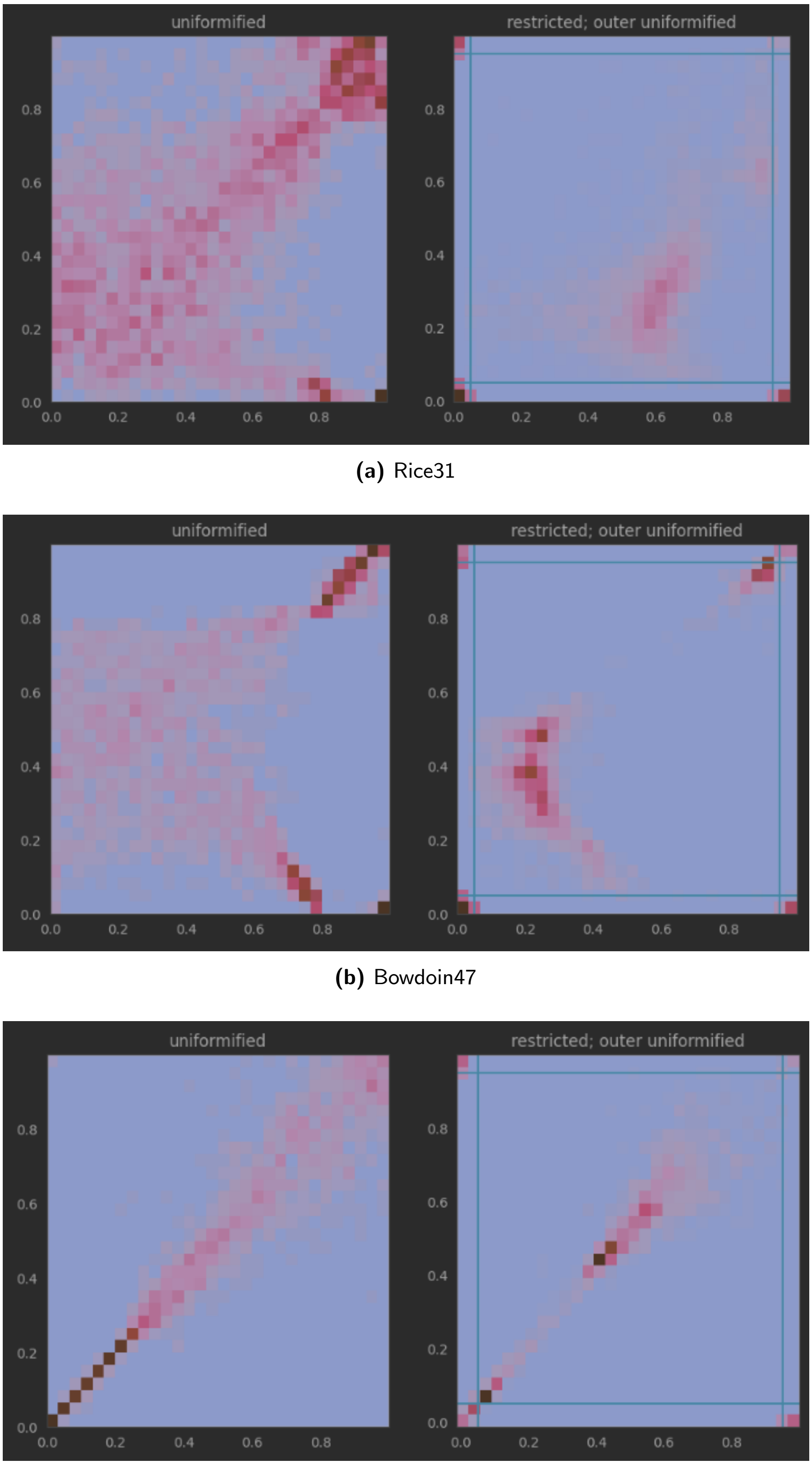
(a) 2d GIRG non-uniformed

(b) 2d GIRG uniformed

(c) socfb-Amherst41 non-uniformed

(d) socfb-Amherst41 uniformed

Figure 6.3: Diffusion map scatter plot of the first two extracted coordinates, with and without using an additional uniform square remapping



(a) Rice31

(b) Bowdoin47

(c) Simmons81

Figure 6.4: two methods for rescaling/shifting diffusion maps into the cube - done here for d=2 truncations. The blue lines for the restricted version show the border at which points are outer uniformified.

A final issue with diffusion maps is that points seem to end up concentrated in corners/edges. My hypothesis is that this is because e.g. on a 1D line segment, it's hard to distinguish somewhat left and very left points - in the end the diffusion cloud bias is still just left leaning. Not sure how much of a problem this is.

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 $x \leftarrow \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$ as most nodes will end up very tightly packed. Instead we 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. 6.4

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_t
```

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

Chapter 7

MCMC

7.1 Introduction

The binary classification framework is designed to compare the similarity of two distributions of datapoints. Each graph is a datapoint, and the classifier compares graphs only on higher level features meant to distinguish distributions of graphs, not individual graphs. In particuular these features are node permutation invariant - such as average degree, effective diameter and so on.

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 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, and positions x_u.

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. Instead we try a Markov-Chain MonteCarlo (MCMC) approach to sample a set of locations $\{x_u\}$ from the posterior distribution.

7.2 Formulation

MCMC is a method in the bayesian framework of a parametric generative model. Parameter θ has prior $p(\theta)$. Datapoint $z \sim p(z|\theta)$. 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 with states $\theta \in \Theta$, and transition probabilities derived from $Q(\theta)$. In particular we will use a Metropolis-Hastings style markov chain. 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 $\theta^{j_1}, \theta^{j_2}, ...$ 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.

I think we do a Gibb's sampling approach. This means both 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 $p(G|\theta) = \prod u, v \in V; u \neq vp(e(u,v)|w_u, w_v, x_u, x_v, \alpha, c)$. This lends well to Gibb's sampling - we need concentrate only on $Q_u(\theta_u) = \prod v \neq up(e(u,v)|...)$.

Key elements of the MCMC approach are 1. burn in time, and 2. proposal distribution.

Burn in time can be quite long. One key component is a good initialisition. With the GIRG model, the natural initialisation for x_u would be to follow the prior $x_u \sim U([0,1]^d)$. Instead we use an initialisation based on d-truncated diffusion map from the real graph connectivity. We also try to optimise our code and use multiprocessing to speed up the random walk.

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, this is

$$A(x'_u, x_u) = \min\left(1, \frac{p_{prop}(x_u|x'_u)p(G|x'_u)p(x'_u)}{p_{prop}(x'_u|x_u)p(G|x_u)p(x_u)}\right) \tag{7.1}$$

$$= \min\left(1, \frac{p(G|x'_u)}{p(G|x_u)}\right) \tag{7.2}$$

$$\text{as } p_{prop} \text{ is symmetric and } p(x) = 1\ \forall x \tag{7.3}$$

7.3 Model Comparison

We seek to show that the GIRG model does a better job at fitting the facebook graphs than the ChungLu model, which is the natural comparison point as essentially a GIRG without geometry. We would also like to determine between 1D, 2D etc. GIRGs what is the best dimensionality.

7.3.1 Likelihood comparison of GIRG vs CL fit to real graph

As a first sanity check, we can sample a $\hat{\theta}_{\text{GIRG}}$ (not MLE but still pretty good) as from the posterior $\theta_{\text{GIRG}}|G$ with MCMC, and more simply fit a $\hat{\theta}_{CL}$. Given that the 1D GIRG is a more parametrised model, it would be total failure if it didn't reproduce the given graph better. For most of the facebook graphs however, $p(G|\hat{\theta}_{\text{GIRG}}, \mathscr{G}_{\text{GIRG}}) < p(G|\hat{\theta}_{CL}, \mathscr{G}_{CL})$! The GIRG model is far too confident about edges, giving $p_{uv} \approxeq 1$ for small $\|x_u - x_v\|$ and $p_{uv} \approxeq 0$ for large $\|x_u - x_v\|$. Hence a mistake on a few edges can lead to a large penalisation in likelihood. The ChungLu 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:

$$p_{uv} = (1-f)\min\left\{1, c\left(\frac{w_u w_v / W}{\|x_u - x_v\|^d}\right)^\alpha\right\} \tag{7.4}$$

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 ChungLu model - i.e. let

$$p_{uv} = \eta\ p_{uv}^{CL} + (1-\eta)\ p_{uv}^{GIRG} \tag{7.5}$$

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 ChungLu 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 ChungLu 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 ChungLu model: $p(G|\hat{\theta}_{GIRG}, \mathcal{G}_{GIRG}) > p(G|\hat{\theta}_{CL}, \mathcal{G}_{CL})$. 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 $f = 0.3, \eta = 0.5$.

7.3.2 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 classification confusion matrix

$$\begin{array}{|c|c|} \hline TP & FN \\ \hline FP & TN \\ \hline \end{array} \tag{7.6}$$

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. 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.

The results of running the MCMC process is to achieve PEC of about??

TODO rerun MCMC for larger graphs for longer. We didn't seem to be converging. I.e. it seems like we need a greater than linear iteration scaling to converge??

real graph	PEC	PEC CL	Number iterations run
socfb-Caltech36-1d.pkl	0.218726	0.057174	134400
socfb-Reed98-1d.pkl	0.172603	0.039337	173600
socfb-Haverford76-1d.pkl	0.210710	0.058752	257600
socfb-Simmons81-1d.pkl	0.168839	0.030439	268800
socfb-Swarthmore42-1d.pkl	0.181657	0.044276	296800
socfb-Amherst41-1d.pkl	0.181718	0.033742	403200
socfb-Bowdoin47-1d.pkl	0.132771	0.033619	403200
socfb-Hamilton46-1d.pkl	0.147926	0.036735	414400
socfb-Trinity100-1d.pkl	0.140800	0.033099	470400
socfb-USFCA72-1d.pkl	0.120164	0.017381	481600
socfb-Williams40-1d.pkl	0.127645	0.029190	498400
socfb-Oberlin44-1d.pkl	0.112043	0.020820	526400
socfb-Smith60-1d.pkl	0.088302	0.021486	532000

7.3.3 point initalisation and failure rate

Initially I ran MCMC using uniformified diffusion map initialisation, and failure rate 0.0. The MCMC would converge, but actually take longer than necessary, as the "good" initialisation was being undone in favour of quite spaced out points.

The evidence for this is shown in some plots of points over time:

Run long enough, the MCMC's points converge to the cube:

LL maximum for 1D	**LL maximum for 2D**
socfb-Amherst41	socfb-Bowdoin47
socfb-Oberlin44	socfb-Caltech36
socfb-Reed98	socfb-Colgate88
socfb-Swarthmore42	socfb-Hamilton46
socfb-Wellesley22	socfb-Haverford76
	socfb-Middlebury45
	socfb-Pepperdine86
	socfb-Santa74
	socfb-Simmons81
	socfb-Smith60
	socfb-Trinity100
	socfb-USFCA72
	socfb-Vassar85
	socfb-Williams40

Table 7.1: Your Caption

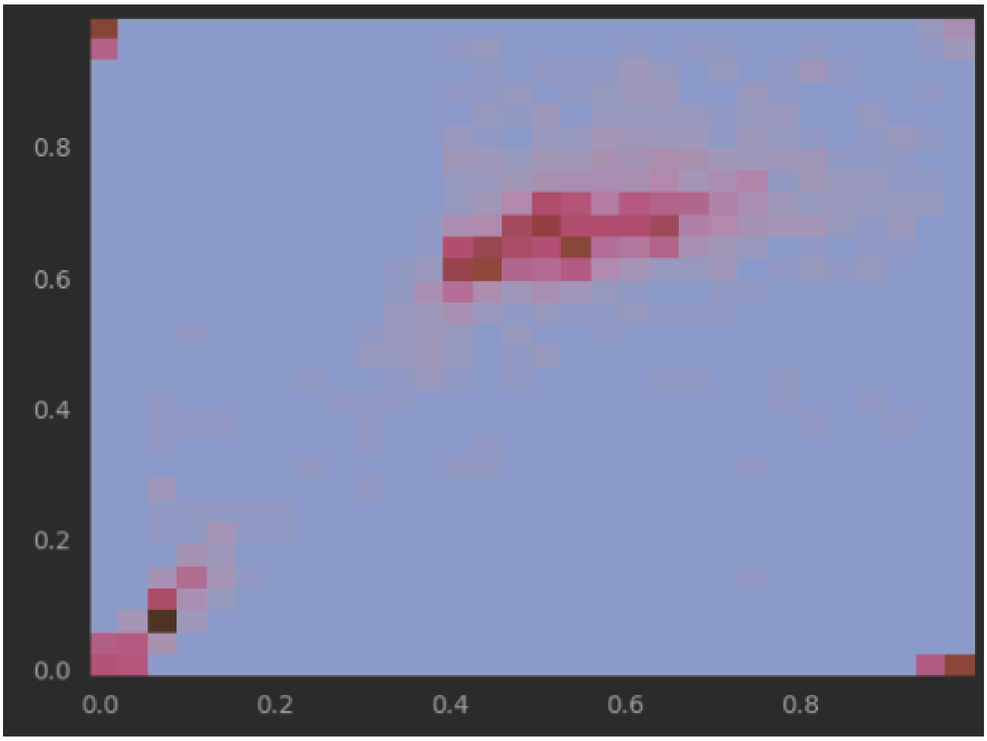
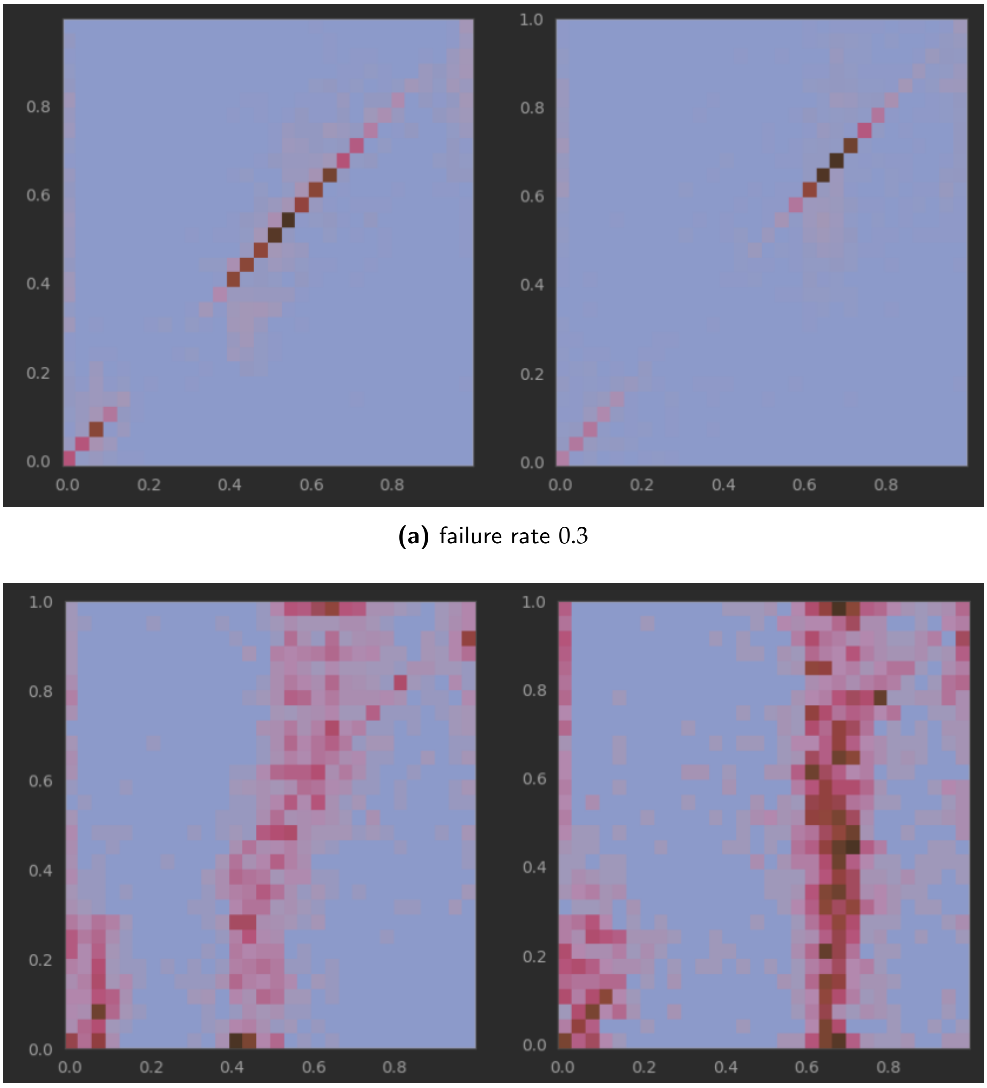


Figure 7.1: 0.05 quantile restricted and uniformed at the edges diffusion map initialisation of MCMC points for socfb-Amherst41



(a) failure rate 0.3

(b) failure rate 0.0 (aka no failure rate)

Figure 7.2: MCMC iterated points scatter plot against initialisation, with/without failure rate

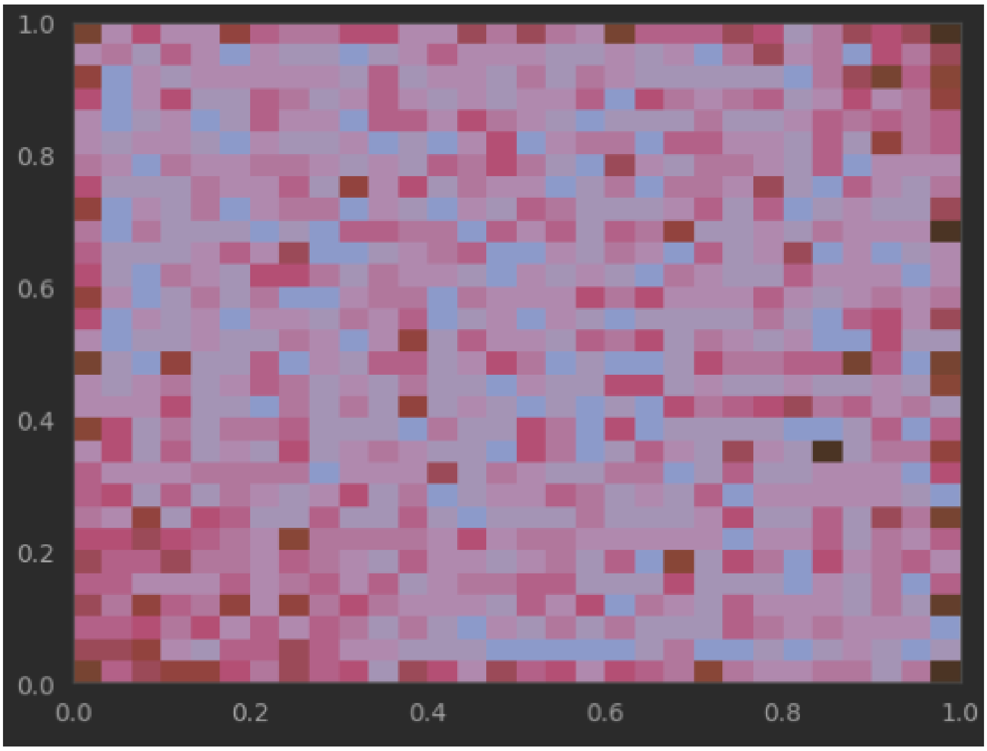
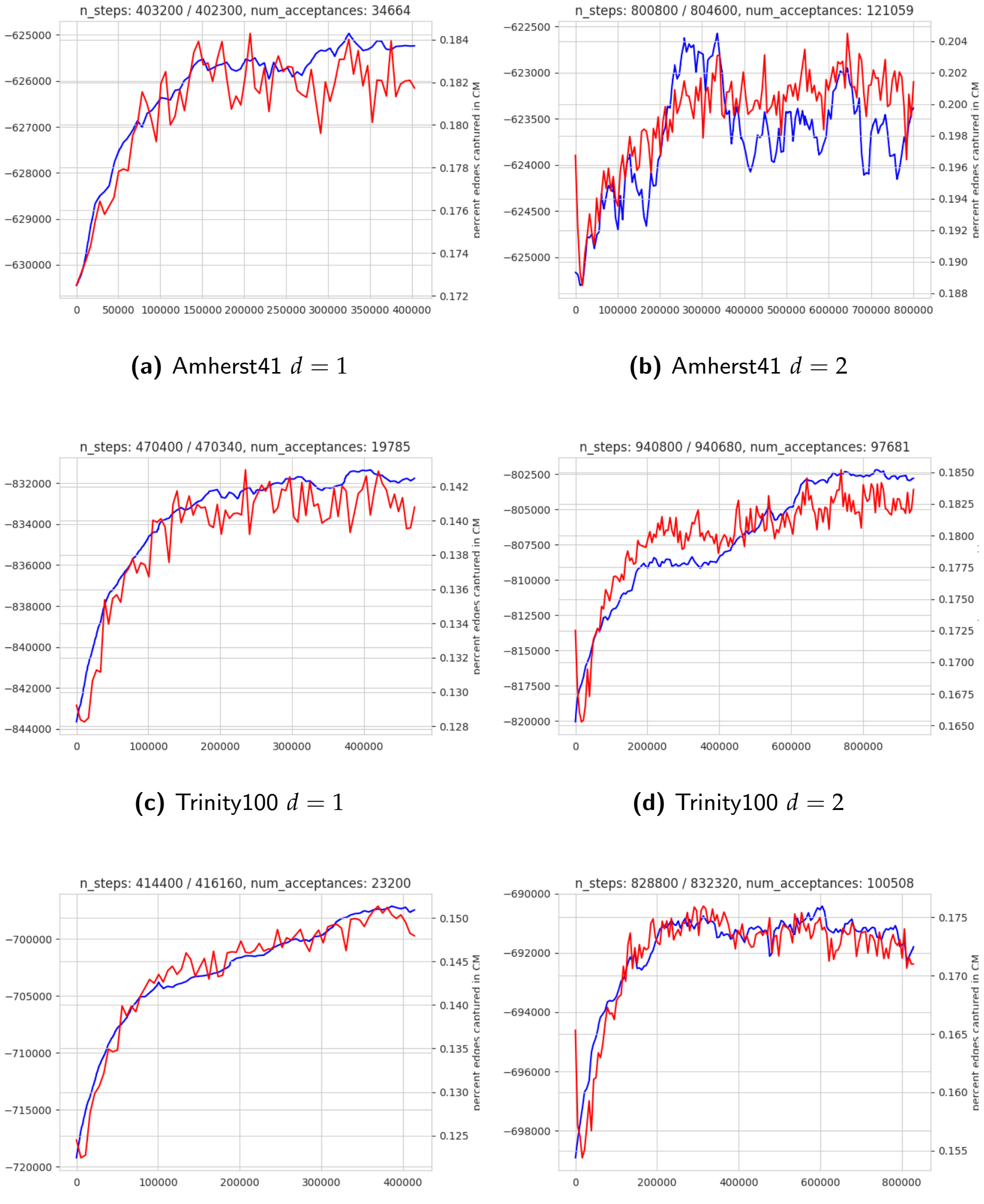


Figure 7.3: failure rate 0.0 run until convergence for 2D GIRG: points 2D histogram



(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 7.4: MCMC runs for socfb-Amherst41, sofcb-Trinity100, socfb-Hamilton46 with failure rate 0.3 and ChungLu Mixin rate 0.5 Log likelihood wise, Amherst 1D GIRG does best; for Bowdoin it's 2D GIRG

Chapter 8

Graph Kernels

8.1 Bayes Factor

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. $M_1 = \mathscr{G}_{1D \text{ GIRG}}$ and $M_2 = \mathscr{G}_{CL}$. D is a single graph instance G, or alternatively a whole dataset of our 100 facebook graphs, assuming that they all come from the same generative model family. We have some prior of $p(M_1)$ vs $p(M_2)$, e.g. 50 : 50, or for the possible different dimensional GIRGS some kind of decaying $p(1D \text{ GIRG}) > p(2D \text{ GIRG}) > p(3D \text{ GIRG}) > \dots$. Putting this together,

$$p(M_1|D) = \frac{p(D|M_1)p(M_1)}{p(D)} \implies \frac{p(M_1|D)}{p(M_2|D)} = \frac{p(D|M_1)p(M_1)}{p(D|M_2)p(M_2)} \tag{8.1}$$

Hence model selection is done with the ratio $\frac{p(M_1|D)}{p(M_2|D)}$; this is called the Bayes Factor. We will ignore the priors $p(M_1), p(M_2)$ for now, and focus on the likelihoods $p(D|M_1), p(D|M_2)$.

More rigorous bayesian model comparison In our case if M_1 is a 1D GIRG, and we've decided to fix e.g. $\alpha, e, \{w_u\}_{u \in V}$, 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,

$$p(D|M_1) = p(G|\mathscr{G}_{\text{d-GIRG}}) = \int_\theta p(G|\theta, \mathscr{G}_{\text{d-GIRG}})p(\theta|\mathscr{G}_{\text{d-GIRG}})d\theta \tag{8.2}$$

The simplified notation is dropping the $|M_1$ as we've 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 for sure. This is an abuse of notation - normally $p(G|\theta)$ is not a sum over permutations, but

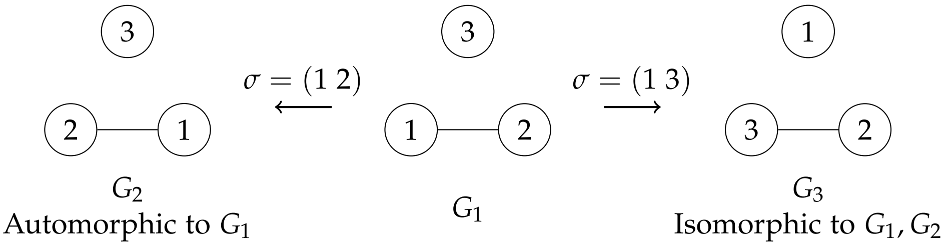


Figure 8.1: Example automorphic/isomorphic graphs

rather the probability of generating this exact graph. This will have to be contextually apparent.

Hence our simple MonteCarlo computation of $p(G|\theta, \text{GIRG}) < p(G|CL)$ which was sad, is actually not quite valid. So Yay! We can still hope that the GIRG model is better than ChungLu, or at least bayesianly more likely.

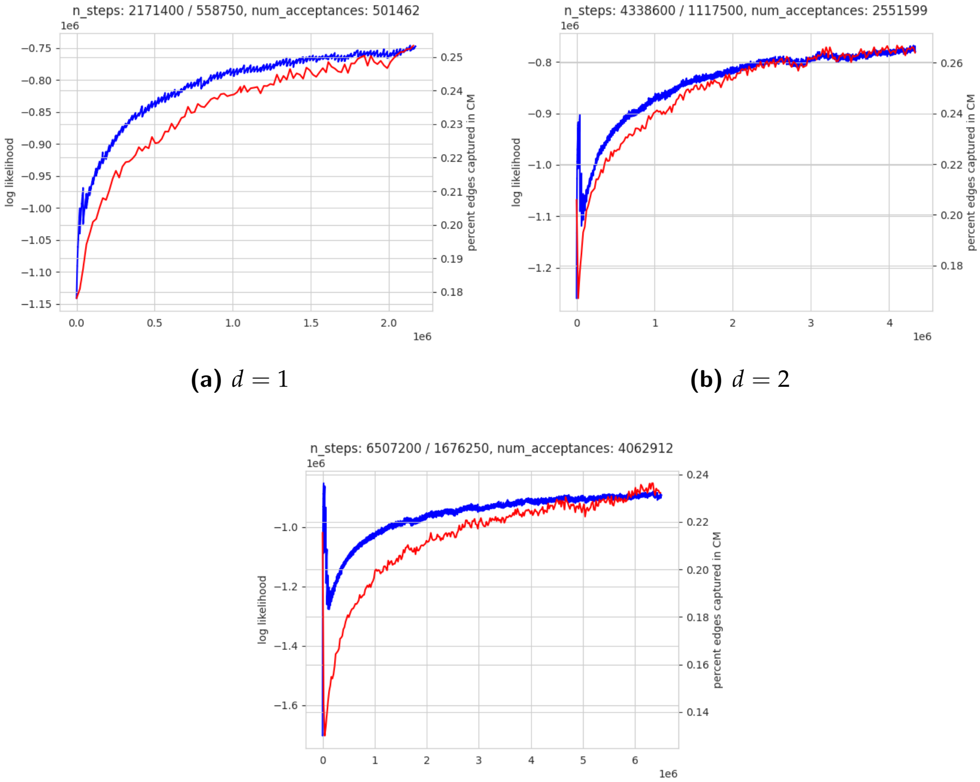
To be concrete, let's say $G = (1,2),(2,3),(2,4),(2,5)$. Then we say G is isomorphic to G' with permutation $\sigma = (1,4)$ ($G \overset{\sigma}{\to} G'$) where $G' = (4,2),(2,3),(2,1),(2,5)$. So the compuation is actually sample θ, then for each $\sigma \in S_n$ create G' where G is isomorphic to G' with permutation *sigma* ($G \overset{\sigma}{\to} G'$), and calculate $p(G'|\theta)$. Add these up and take the average. Finally repeat for more different θ samples and take an outer average.

To be concrete, take an example graph $G = (V, E) = (\{1,2,3\}, \{(1,2)\})$. 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. 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 $1, 2$.

So the setup is that with G with 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.

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.

Unfortunately computing the $n!$ sized $\sum_\sigma p(G' \overset{\sigma}{\cong} G|\theta)p(\sigma)$ is infeasible for



(a) $d = 1$

(b) $d = 2$

(c) $d = 3$

Figure 8.2: MCMC runs for socfb-Amherst41, without failure rate

all but tiny graphs.

There are some alternatives. One is to use a graph similarity kernel which compares two graphs G, H, giving some measure of how isomorphically similar they are. 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 $\mu(G) = E_{G' \sim \mathscr{G}_{\text{d-GIRG}}}[k(G, G')]$.

So when we say $p(G|\theta)$, we actually mean the probability of getting a graph G' which is isomorphic to G, given parameters θ.

We see in fig. 8.2 that Bayes Factor model comparison wise, 1D GIRGs are superior, even though according to edge accuracy, 2D GIRGs achieve a slight edge.

8.2 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 combination 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. As in the proposed paradigm in the previous section, 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. This is simply done by comparing $k(G, G_1)$ and $k(G, G_2)$ - the higher of the two is the more similar.

Graph Kernels have many uses, but we can use them in particular to help with bayesian model comparison, solving our issue of graph permutations.

A survey on graph kernels https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0195-3

We saw the equation $\mu(G) = E_{G' \sim \mathcal{G}_{\text{d-GIRG}}} [k(G, G')]$ in the previous chapter. We can Monte Carlo sample this expectation to get an estimate.

8.2.1 Random Walk Kernel; Weisfeiler-Lehman Kernel

There are many graph kernels to choose from. However given the relatively large size of our graphs, we opted to mainly test the Weisfeiler-Lehman Kernel. It has the benefit of running 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.

8.2.2 Experiments

we do the comparison

Cube Similarity Plots: We fit alpha, c for a uniform cube GIRG for dimensions d=1-3, produce a graph, and look at similarity with real graph.

Appendix A

Dummy Appendix

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

Bibliography

[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.

[Bringhurst, 1996] Bringhurst, R. (1996). *The Elements of Typographic Style*. Hartley & Marks.

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

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

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

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

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

Title of work (in block letters):

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

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

With my signature I confirm that

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

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

Place, date **Signature(s)**

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