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:

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

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:

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

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)

Contents

4.1 GGMs

[Bläsius et al., 2018] compares various generative graph models: Erdos-Renyi (ER), Barabasi-Albert (BA) preferential attachement graphs, Chung-Lu and Hyperbolic Random 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 there are n nodes: $V = [n]$, and each potential edge (u, v) exists iid with probability p. We write this as $P(u \sim v) = p$.

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 $\hat{p} = |E|/\binom{n}{2}$ 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 for instance, 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 (full definition to come in section 5.1): for example we cannot even easily integrate over all possible point locations in the geometric space:

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

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

4.1.3 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 8 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, 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.

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 $\mathscr{G}$, but with differing parameters: $G_1 \sim \mathscr{G}(\theta_1^*), ..., G_k \sim \mathscr{G}(\theta_k^*)$. 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 $\boldsymbol{f}^{\textbf{class}}(G)$ 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 $\boldsymbol{f}^{\textbf{class}}(G_i)$ and $\boldsymbol{f}^{\textbf{class}}(G_i')$. 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 $\tilde{\mathscr{G}}$.

Chapter 5

GIRG generation

Contents

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

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

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 $W = \sum_{u\in V} w_u$ ensures that for a node u, as n increases (even $n \to \infty$), though it will have more possible other nodes to connect to, its expected degree won't change.

5.1.1 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).

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

This is a heavy tailed distribution (heavier than exponentially decaying tails). $\tau > 2$ is important to ensure that $E[\mathrm{w}] = \Theta(1)$. 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.

5.1.2 Geometry

This edge probability has a geometric factor from the distance $r_{uv} = \|x_u - x_v\|_\infty$, which is taken to the dth power so that the number of edges is consistent across different values of d. This inversely scales the probabilities so that nearby points (small r_{uv}) are more likely to have an edge than those further apart. This 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.

The geometric space χ, and the distance function $\|\cdot\|$ can vary a great deal without affecting the key properties of GIRGs. $\chi = \mathbb{T}^d$ the d-dim 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-dim cube $\chi = [0,1]^d$ can be more realistic; however 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.

[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...

Taking $\|\cdot\| = \|\cdot\|_\infty$ is also useful, but can be replaced equivalently by euclidean or other norms. The minimum component distance $\|x\|_{\text{mcd}} = \min_i |x_i|$ is not a norm, but still 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.

$\alpha \in (1, \infty]$ is another parameter affecting the geometry. The edge probability formular distinguishes **short** edges where $r_{uv} < (w_u w_v / W)^{1/d}$, which have $p_{uv} = \Theta(1)$, 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 speeds the decay, and $\alpha = \infty$ is essentially a sharp cutoff to $p_{uv} = 0$ for long edges.

5.1.3 Similarity to Chung-Lu

A key property of the GIRG model is that despite the influence of geometry on edge probabilities, you can "ignore" this factor and get a specific simplified model. So 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 $E_{x_v}[p_{uv} | x_u, w_u, w_v] = \Theta(\min\{1, \frac{w_u w_v}{n}\})$, which is precisely the Chung-Lu GGM.

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.

5.2 Fitting GIRGs for Blasius evaluation framework

As explained in chapter 4, we wish to compare GIRGs with other GGMs for their ability to fit a dataset of real social network graphs. In section 4.1.2, we introduced the ABC method for fitting a GGM to a particular real graph instance $G = (V, E)$, which we will use for GIRGs, similarly to how [Bläsius et al., 2018] fits Hyperbolic Random Graphs.

A first important step for ABC is the ability to generate GIRGs, i.e. sample $G \sim \mathscr{G}_{\text{GIRG}}(\theta)$.

5.2.1 Blasius C++ GIRG generation

We started by using the C++ implementation of GIRG generation in [Bläsius et al., 2022]. Their GIRG formulation uses a toroidal geometry $\chi = \mathbb{T}^d$, with edge probabilities

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

i.e. no outer Θ which gave us more flexibility in the general GIRG definition, instead just one fixed inner constant c. This still falls under the wider GIRG definition as p_{uv} always lies in the interval

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

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

They implement the algorithm in [Bringmann et al., 2019], which they claim to have linear runtime. A GIRG, having power law weighted nodes, has $E[d_u] \propto E[w_u] = \Theta(1)$, so with high probability has $\Theta(n)$ number of edges, i.e. linear in number of nodes. Hence a linear runtime algorithm ($O(n)$) is optimal, as even an oracle must write out the list of edges sequentially in $\Theta(n)$ time.

We also implemented our own GIRG generation code in python for ease of modification (see many different GIRG variants to come), with the same edge probabilities eq. (5.2). This is done more simply with a $O(n^2)$ runtime, as well as $O(n^2)$ 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 or not

Unfortunately the Blasius algorithm scales quite badly in dimension d, such that for larger graphs of with $d \geq 4$, the python implementation was preferrable.

ABC recap Hence our full power law weighted torus GIRG parametrisation is $\theta = (n, d, c, \alpha, \tau)$

We fit the model to a certain real graph instance $G = (V, E)$ in a few steps, using ABC as introduced in section 4.1.2, and different distance metrics $d(G, G')$ for each subcomponent of θ. All we need to do is be able to propose values of θ and generate a graph $G' \sim \mathscr{G}_{\text{GIRG}}(\theta)$ from them.

5.2.2 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 5.1. 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 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.

5.2.3 Fitting power law distribution exponent τ for node weight sampling

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

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).

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 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 $dd(x) \propto x^{-\tau}$.

A brief sketch of why this occurs is by treating node degrees as roughly iid, $dd(x) \overset{n\to\infty}{\to} P(d_u = x)$. d_u given w_u is roughly a binomial distribution binomial$(n-1, \Theta(w_u/n))$, and we'd like to calculate $\int \Theta(w^{-\tau}) P(d_u = x | w_u = w) dw$. The binomial distribution of mean and variance w is sharply peaked around its mean: if $(x - w) > w^{1/2+\varepsilon}$ then $P(d_u = x | w_u = w) \approx 0$.

This means that for sufficiently large x, we can focus on just the subsection

of the integral $\int_{x-x^{1/2+\varepsilon}}^{x+x^{1/2+\varepsilon}}$. In this subsection, $\Theta(w^{-\tau}) \approxeq \Theta(x^{-\tau})$ (not much appreciable difference), and we can approximate the binomial distribution as a gaussian with mean and variance w. This has pdf $\frac{1}{\sqrt{2\pi w}} e^{-(x-w)^2/2w}$ which we can approximate as $\frac{1}{\sqrt{2\pi x}} e^{-(w-x)^2/2x}$.

Finally we evaluate

$$\int_{x-x^{1/2+\varepsilon}}^{x+x^{1/2+\varepsilon}} \frac{1}{\sqrt{2\pi x}} e^{-(w-x)^2/2x}\, \Theta(x^{-\tau}) dw = \Theta(x^{-\tau}) \tag{5.4}$$

to get.

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 first finding a lower bound $x_{\min}$ for the power law behaviour, and then fitting τ to the tail $x > x_{\min}$.

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 empririal degree distribution.

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

5.2.4 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 $w_u = x_{\min}$, rather a more natural curve like in fig. 5.1.

An easy improvement for fitting a specific real graph is to take the sequence of node degrees as weights [2]. 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). This doesn't make a fair / like for like comparison, so we cover all bases by having both weight copied and power law fit GIRGs, as well as adding in power law fit Chung-Lu. ER and BA GGMs are of course just fit on the courser metric of overall graph average degree.

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

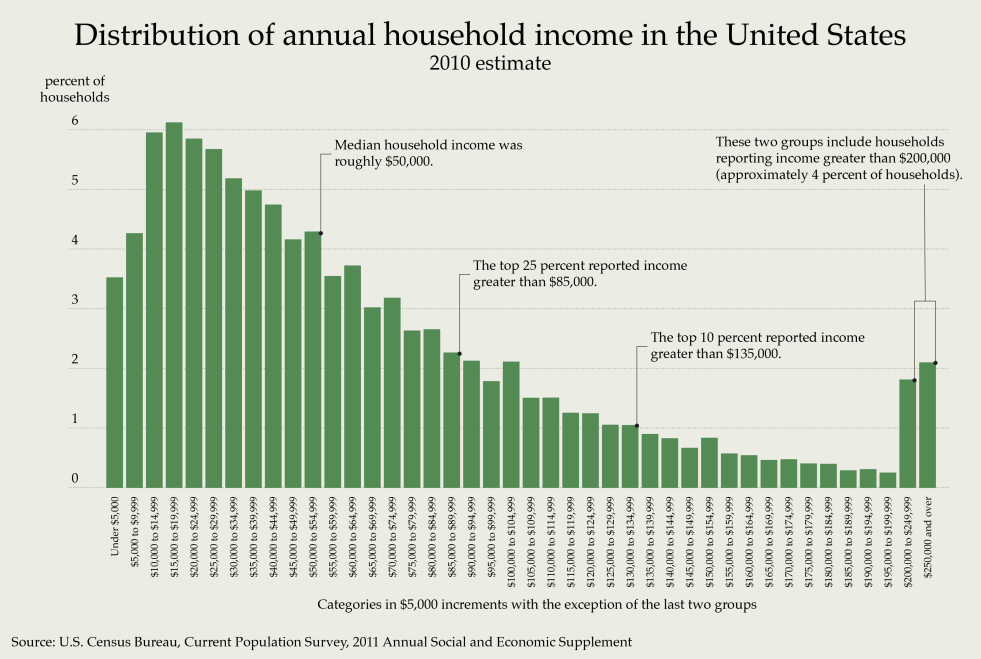


Figure 5.1: Example of a more natural weight distribution with a power law tail but not for smaller weights.

Comparing weight copied Chung-Lu with weight copied GIRGs was additionally interesting 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%).

5.2.5 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 more geometric, clustered) edges. We hence follow [Bläsius et al., 2018] by fitting α with the clustering coefficient. Unfortunately in our experiments we used the (average) local clustering coefficient (LCC) to fit our GIRGs and Hyperbolic Random Graphs,

$$\mathrm{LCC}(G) := \frac{1}{|V|} \sum_{u \in V} \frac{|\{v, v' \in \Gamma(u) : v \sim v'\}|}{\binom{|\Gamma(u)|}{2}} \tag{5.5}$$

instead of the global clustering coefficient as in [Bläsius et al., 2018]. The global clustering coefficient, the fraction of total "V" shapes that indeed are

completed as full triangles is likely a better metric than the LCC. Luckily in practice they generally differ by less than 1%.

Unfortunately c, 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 alternatively maximise $c \leftarrow \hat{c}_1$; $\alpha \leftarrow \hat{\alpha}_1$; $c \leftarrow \hat{c}_2$; $\alpha \leftarrow \hat{\alpha}_2$; ....

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 $|\text{LCC}(G') - \text{LCC}(G)|^2$. To both propose the next candidate α_i and eventually accept the final best $\hat{\alpha}_i$.

[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 $\mathbb{E}[\overline{deg}]$ of the GIRG.

$$f(c) = cA + c^{1/\alpha}B \tag{5.6}$$

Hence we just have to numerically solve eq. (5.6) for $\hat{c} : f(\hat{c}) = \overline{deg}$ in order to obtain our desired average degree $\overline{deg} = 2|E|/|V|$.

The expected average degree formula of eq. (5.6) 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.

Derivation of GIRG expected average degree formula The volume formulation of a GIRG presents a useful generalisation:

$$p_{uv}(r) = \min\left\{1, c\left(\frac{w_u w_v / W}{Vol(r_{uv})}\right)^{\alpha}\right\}; \quad \rho_{uv} = \frac{w_u w_v / W}{Vol(r_{uv})} \tag{5.7}$$

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 must be symmetric to make sense. For example in the ∞-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. So we'll derive $E_r[p_{uv}(r)]$ for a general volume function $Vol(r)$, and fixed weight sequence $(w_u)_{u \in V}$ and use this to recover eq. (5.6) via $E[\overline{deg}] = \sum_{u \in V} \frac{1}{n} \sum_{v \in V} E_r[p_{uv}(r)]$.

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 $\hat{r} : Vol(\hat{r}) = \frac{w_u w_v}{W} c^{1/\alpha}$ as the boundary radius 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:

$$E_r[p_{uv}(r)] = \int_0^{Vol(\hat{r})} dVol + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} p_{uv}(Vol)dVol \tag{5.8}$$

$$= Vol(\hat{r}) + \int_{Vol(\hat{r})}^{Vol(\mathbb{T})} c\left(\frac{w_u w_v}{W}\right)^{\alpha} Vol^{-\alpha} dVol \tag{5.9}$$

$$= Vol(\hat{r}) + c\left(\frac{w_u w_v}{W}\right)^{\alpha} \frac{1}{\alpha - 1}\left[\left(\frac{w_u w_v}{W}\right)^{1-\alpha} c^{\frac{1-\alpha}{\alpha}} - 1\right] \tag{5.10}$$

$$= c^{1/\alpha}\left(\frac{w_u w_v}{W}\right)\left[1 + \frac{1}{\alpha - 1}\right] - \frac{c}{\alpha - 1}\left(\frac{w_u w_v}{W}\right)^{\alpha} \tag{5.11}$$

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 - and $\Theta(\frac{w_u w_v}{W})$ as promised!

Following [Bläsius et al., 2022] Appendix, the formula just needs a correction for pairs u, v such that $Vol(\hat{r}) = c^{1/\alpha}\left(\frac{w_u w_v}{n}\right) > 1$, wherein the second integral is unnecessary, and the first 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.

5.3 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 (only whole planet scale geographic locations look more torus-esque). 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 a toroidal-esque quantity 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.

5.3.1 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. Where before we could write e.g. $Vol(r_{uv}) = (2\|x_u - x_v\|)_\infty^d$, now in cube geometry $Vol(B_{r_{uv}}(u)) = Vol(B_{r_{uv}}(u)) \cap [0,1]^d$, 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 $r_{uv}^C \geq r_{uv}^{\mathbb{T}}$, with most pairs having $r_{uv}^C = r_{uv}^{\mathbb{T}}$. We can say that a GIRG in cube geometry is stochastically dominated by its torus geometry counterpart (and 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, have the same desire to make friends (modulo their inhomogeneous weights as that is the literal introversion / extroversion factor in the GIRG model). Thus if we were modelling directed edges, we could have equality in $E_r[p_{u\to v}(r)]$ in all GIRGs of differing distance metric, number of dimensions and torus / cube geometry by using volume based formulations. 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, it may send out the normal volume based amount of friendship invitations ($u \to v$), but if most of these go to nodes v nearer the centre of the cube, fewer will reciprocate ($v \to u$). In the torus geometry (all weights being equal), every u would have the same expected number of outgoing edges as incoming. The volume based cube GIRG formulation would still have higher average degree than the distance based version, but lower than the torus.

5.3.2 Cube GIRG generation - coupling algorithm

We use the distance based cube GIRG formulation, which permits a simple coupling based generation algorithm off of an already generated Torus GIRG, shown in algorithm 1. This runs in $O(n)$ extra time on top of the initial Torus GIRG generation.

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 α. Our distance based Cube GIRGs have fewer edges in expectation than their toroidal counterparts, and no nice formula for the

Algorithm 1 Generate Cube GIRG from Torus via coupling

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

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

5.4 Min GIRGs

In section 5.1 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. $\|x - y\| = \|z\| = \min(z_1, \max(z_2, z_3), \max(z_4, z_5))$. 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.

5.5 Blasius classification results

As introduced in section 4.1.4, 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. Higher classification accuracies, e.g. 99%, denote that the GGM generates are readily distinguishable from the real graphs, and hence the GGM is a poor / unrealistic fit; 50% accuracy is the gold standard of real/fake indistinguishability on the feature set.

We can see from fig. 5.3 that mimicking the features of real Facebook graphs with generations from a fit GGM is tough. For example in the feature set (n, m, deg), by deg we mean to include 5 numbers per graph: mean node degree; lower, middle (median) and upper quartile node degree, and standard deviation over all node degrees. Our numbers can be compared to Table 2 in [Bläsius et al., 2018], which also includes feature sets with just mean node values, which in some cases makes fooling the classifier much easier[3].

Features like k-cores and comms aren't node level, and are taken over the actual core/community sizes - so are e.g. mean, median etc. community sizes within a graph.

Types of GIRG tested in classification framework The base GIRG type we tried was max norm torus GIRGs for dimension d from 1 to 7. Note only $d = 1, 2, 7$ are shown in fig. 5.3 as the classification numbers follow a trend in dimension.

We also tried mcd torus GIRGs for $d = 2, 3, 4, 5$ ($d = 1$ is equivalent to max norm), as well as min/max mixed torus GIRGs with mixing $1 - 23$, $1 - 234$, $12 - 34$, $1 - 2 - 34$. The latter two are excluded from fig. 5.3 as they had worse classification numbers.

Finally we trained some 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).

Finally we have 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-GIRGs.

(effective) diameter Only the features diam and eff-diam are single numbers per graph - as just one number this is easier to mimick, e.g. the 2d, 3d cube GIRGs get 77% accuracy. cube GIRGs' having a more realistic effective diameter than torus GIRGs could just be a result of torus GIRGs having too small diameters compared to real graphs, and cube GIRGs having larger

[3] We do get similar results to Table 2 when training SVMs with just the mean node features, which can yield lower accuracies in the $50 - 70\%$ range

diameter than toroidal[4]. This does however fit our geometric intuition of the realism of cube geometry over toroidal.

closeness centrality To draw another example, for the n, m, close feature set, 2d GIRGs only achieve 98% accuracy. This is just slightly better than CL, BA, ER, but still seems very bad. However fig. 5.2a 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. I would argue that 2d GIRGs did a pretty good job on replicating closeness in real graphs, so the high accuracy of 98% is a little misleading.

Classification accuracy is a crude metric, but should at least be a point of comparison between GGMs. A drawback of this framework is that the attainable accuracy is also affected by the variety / predictability of a feature set across the real graph dataset - whatever the real graphs are doing, the GGMs have to do too - but more variance is easier to blend in with than a strict pattern. Our GGMs furthermore are much more likely to produce a set of similar graphs following a strict pattern.

GIRGs performance We can compare classification performance between different types of GIRGs, as well as between GIRGs and other GGMs; Chung-Lu acts as a good null hypothesis for GIRGs without geometry[5].

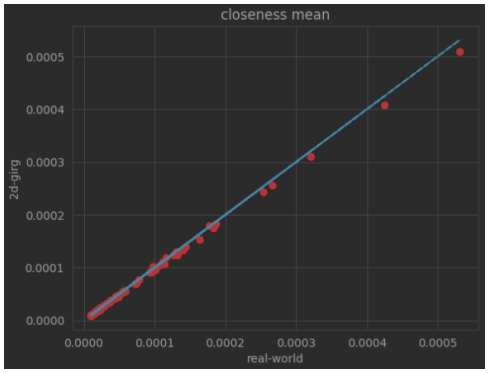
Copy-weight cube GIRGs have a clear cut performance advantage over all other GGMs, only rivalled by copy-weight Chung-Lu. We see, as expected, improvements in diameter and effective diameter over Chung-Lu, and just in effective diameter for non copy-weight GIRGs. Copy-weight GIRGs also have improved LCC performance over copy-weight Chung-Lu, though this is not observed in powerlaw weighted GIRGs - despite explicitly fitting the mean LCC, the wider LCC statistics must be quite distinguishable.

Not-surprisingly copy-weight GGMs have very good degree distribution performance, with Chung-Lu being the best as having no extra geometric modification on the copied degree sequence.

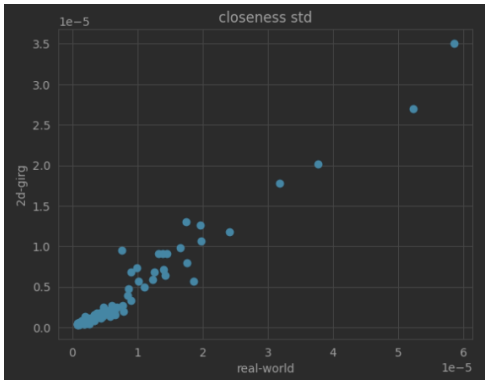
Community sizes and k-core sizes are potentially less consistent features to consider as they have higher variance than node level features.

[4]Looking at the features of real and generated graphs, the effective diameter of the FB graphs range from 3.0 to 4.4; 2d torus GIRGs on average 0.17 less than their real counterparts, and 2d cube GIRGs just 0.04 less.

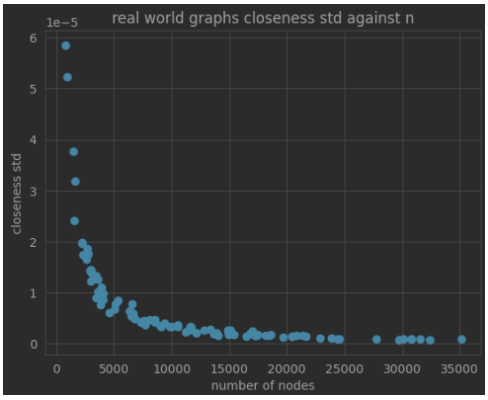
[5]Another potential GGM geometry null hypothesis would have been the configuration model which tries to mimick a degree sequence, like Chung-Lu, but more strictly. Each node is given a set number of "half edges" which gives it a set degree (the degree sequence can be random e.g. power law generated,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



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

are then joined together at random.)

Feature Set	1-ccu	2-ccu	5-ccu	CL-c	1-23	1-234	2-min	5-min	1-cu	2-cu	3-cu	1d	2d	7d	CL	BA	ER	hyper
n, m, betw	82%	84%	95%	95%	99%	99%	98%	100%	99%	100%	100%	99%	99%	99%	99%	100%	100%	99%
n, m, k-core	94%	91%	90%	86%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
n, m, close	92%	85%	84%	90%	99%	100%	100%	97%	98%	97%	95%	98%	98%	98%	99%	99%	100%	97%
n, m, LCC	96%	93%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
n, m, deg	86%	77%	66%	54%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
n, m, Katz	75%	75%	64%	55%	94%	94%	94%	91%	93%	97%	99%	94%	93%	95%	97%	100%	99%	94%
n, m, PR	80%	75%	77%	77%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
n, m, comms	97%	96%	96%	91%	98%	98%	99%	99%	94%	94%	94%	95%	97%	89%	90%	95%	92%	99%
n, m, k-cores	94%	92%	86%	83%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
n, m, diam	96%	97%	97%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
n, m, eff-diam	78%	75%	78%	91%	92%	83%	98%	100%	83%	77%	76%	87%	84%	87%	100%	100%	100%	83%

Figure 5.3: [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	$1 \vee (2 \wedge 3)$ mixed min/max GIRG	k-core	node k-core number
2-min	$1 \vee 2$ 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 5.4: Graph Generative Model abbreviations; feature name abbreviations

Chapter 6

Diffusion Maps

Contents

6.1 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 $\boldsymbol{x}_u \in \chi$.

We saw this already to some extent with the copy-weight GIRGs - where $\hat{w}_u = d_u$ is fit, 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 $\{\boldsymbol{x}_u\}_{u \in V}$ positions.

6.2 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 $(\boldsymbol{x}_u)_{u \in V}$.

6.2.1 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

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

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

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

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

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

$= V\Lambda V^T$ diagonalisation into orthonormal e-vectors

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

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

We use the diagonalisation of S to write M as

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

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

The biorthonormality $\langle \boldsymbol{\phi}_i, \boldsymbol{\psi}_j \rangle = \delta_{ij}$ means that $M\boldsymbol{\phi}_i = \lambda_i \boldsymbol{\phi}_i$ and $M^T\boldsymbol{\psi}_i = \lambda_i \boldsymbol{\psi}_i$. For a diffusion map representation of nodes, we order eigenvalues $\lambda_1 \geq \lambda_2 \geq$ .... Notably the transition matrix satisfies $M\mathbf{1} = \mathbf{1}$ 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

$e_i^T M = \sum_{k=1}^{n} \phi_i(k)\lambda_k^t \boldsymbol{\psi}_k$ diff map of node i after t steps

$i \mapsto (\phi_2(i)\lambda_2^t, ...\phi_{d+1}(i)\lambda_{d+1}^t) = \boldsymbol{\varphi}_t(v_i)$ d-trunc diff map representation

$D_t(v_i, v_j)^2 := \sum_k \frac{1}{d_k}(M_{ik}^t - M_{jk}^t)^2$ *diffusion distance*

$D_t(v_i, v_j)^2 = \left\|\boldsymbol{\varphi}_t(v_i) - \boldsymbol{\varphi}_t(v_j)\right\|^2$ provable equality

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

[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.

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 $\boldsymbol{\pi}_i = \frac{d_i}{\sum_j d_j}$.

6.2.2 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 $\boldsymbol{z}_i \in \mathcal{M}$, 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 6.2.1 used to define the diffusion process.

Naive approach The naive approach is just to use $W = A$. This is a valid methodology akin to taking edge weights 1 or 0 between points $\boldsymbol{z}_i, \boldsymbol{z}_j$ in high dim space if they fall within some $\varepsilon > 0$ distance of each other; this is the "Simple-Minded" variant proposed in [Belkin and Niyogi, 2001]. Furthermore if we wanted to make minimal assumptions on the generative process that produced G, beyond that edges 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 $W_{ij} = e^{-\left\|z_i - z_j\right\|^2/T}$, 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 $\left\|z_i - z_j\right\|^2$ the expected distance between nodes u, v in the HRG - we can do the same for a GIRG: $\mathbb{E}[\|\boldsymbol{x}_u - \boldsymbol{x}_v\|^2]$. This is our best distance guess given our assumption that G is GIRG generated.

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 $\boldsymbol{x}_v - \boldsymbol{x}_u$ 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

$$M_{uv} \leftarrow \frac{M_{uv} d_v^{-\gamma}}{\sum_{v'} M_{uv'} d_{v'}^{-\gamma}}$$
$$M \leftarrow D' M D^{-\gamma} \quad \text{(diagonal } D' \text{ normalises probabilities)}$$

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 $w_v \sim \text{powerlaw}(\tau - 1)$; our modified random walk has $w_v \sim \text{powerlaw}(\tau)$, 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.

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

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

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

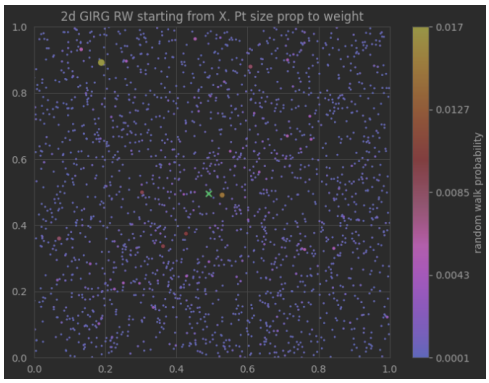
Which is bounded, unlike in the original naive approach where

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

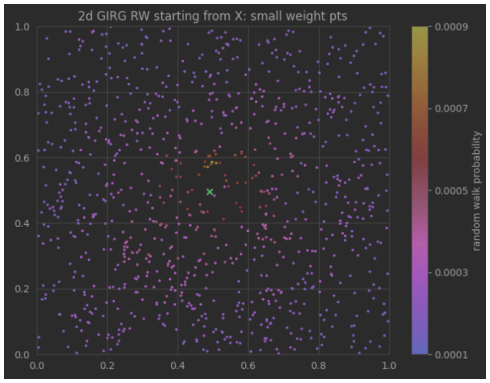
And in our $\gamma = 1$ approach homogeneous random walk approach,

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

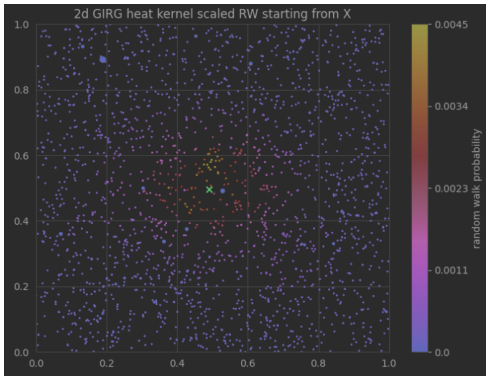
In fig. 6.1 diffusion clouds out of a point are shown for the three different edge weighting approaches. Both non naive versions look more like brownian



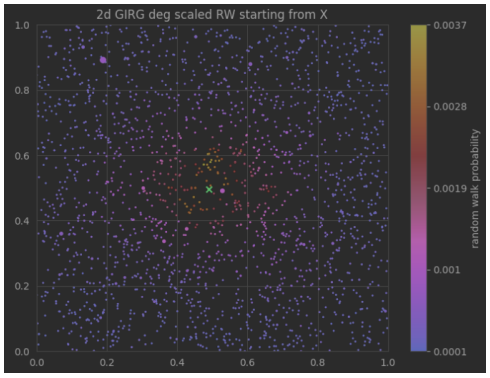
(a) Naive approach: Large weight nodes (drawn with larger radii) get disproportionately more probability mass.



(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 6.1: 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.

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 $\gamma = 1.0$.

6.3 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 6.2.2 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.

6.3.1 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.

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. 6.2 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.

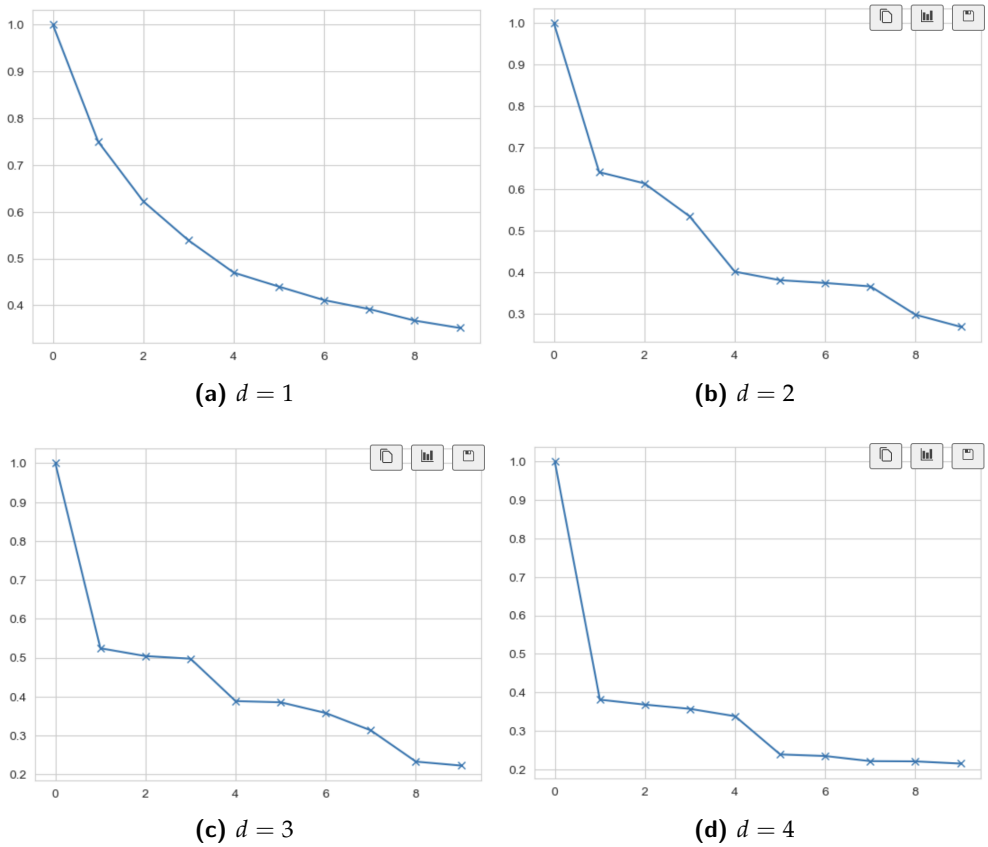
TODO plot some real graph eigenvalue plots?

6.3.2 Rescaling/Shifting Diffusion Maps

We see in fig. 6.4 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 $\boldsymbol{\phi}_1 = k\mathbf{1}$: 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.

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 $(\boldsymbol{x}_u)_i$ with its percentile value compared with other $(\boldsymbol{x}_v)_i$. fig. 6.5 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



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

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

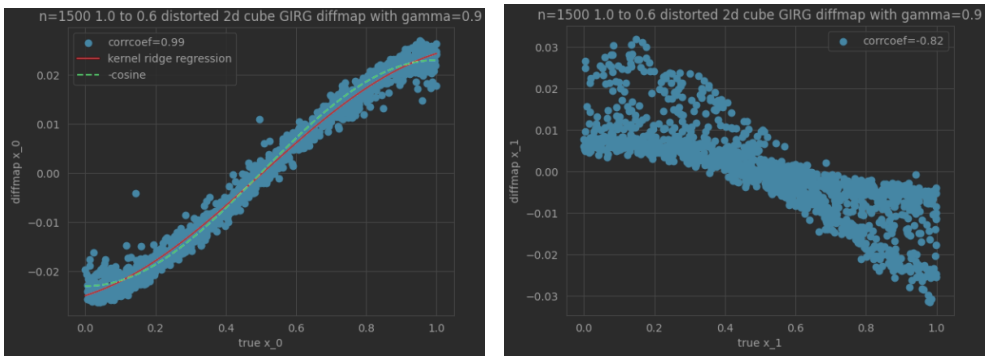
Figure 6.2: Diffusion map eigenvalues (including $\lambda_1 = 1$) for a $n = 2000, \tau = 2.5, \alpha = 1.3$ Cube GIRG with $d = 1, 2, 3, 4$ dimensions.

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. 6.4b, 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 under-



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

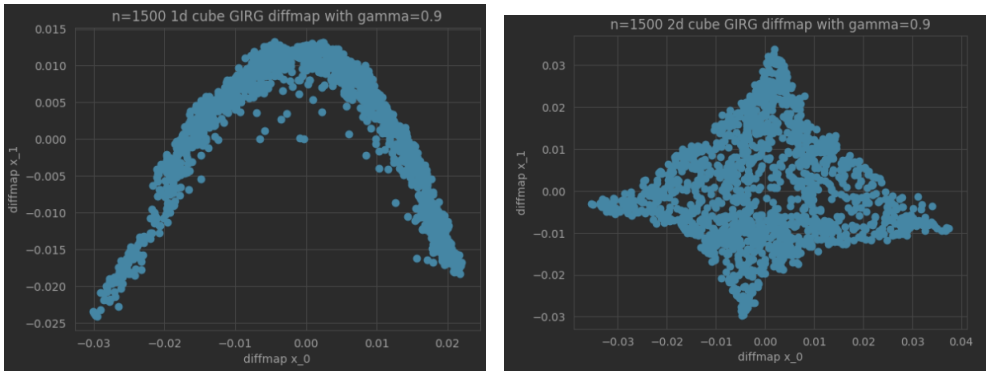
Figure 6.3: Major and minor diff map against real

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

fig. 6.3a 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. 6.2d 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



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

Figure 6.4: Diffusion map scatter plot of the first two extracted coordinates from 1d and 2d GIRGs.

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.

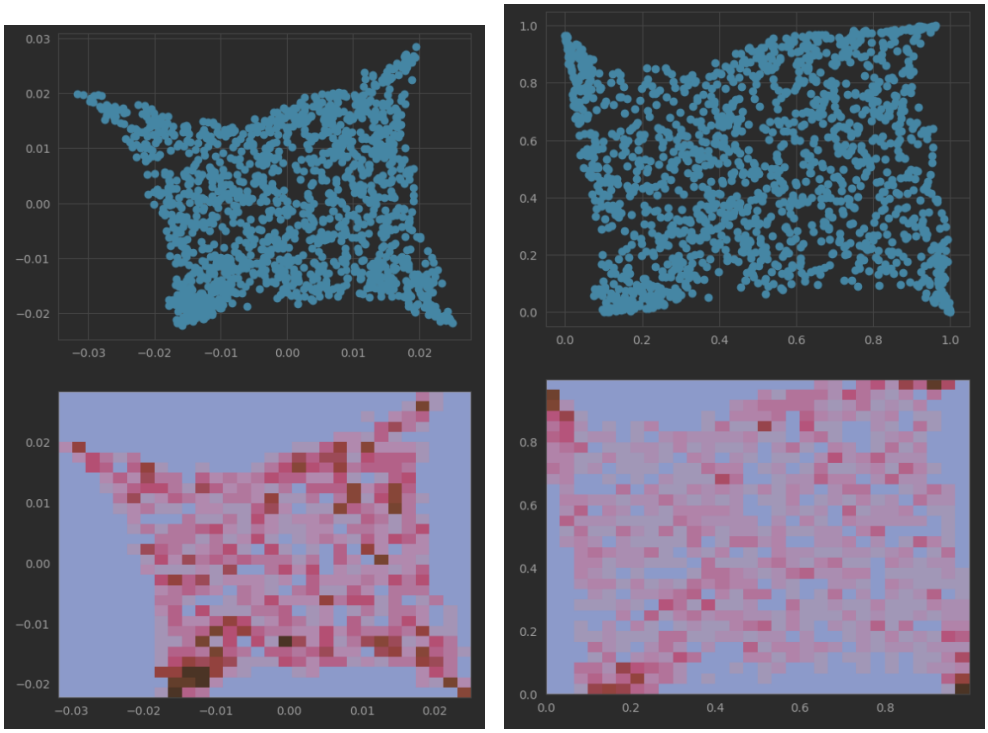
Instead we only rescale the central nodes: those whose joint coordinate-wise percentiles lie in $[5\%, 95\%]^d$. These nodes are linearly scaled to the $[0.05, 0.95]^d$ cube. Finally the outlying nodes are percentile rescaled (non-linearly) to the upper/lower cube margins. This method is shown in comparison for a few graphs in fig. 7.4

TODO

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

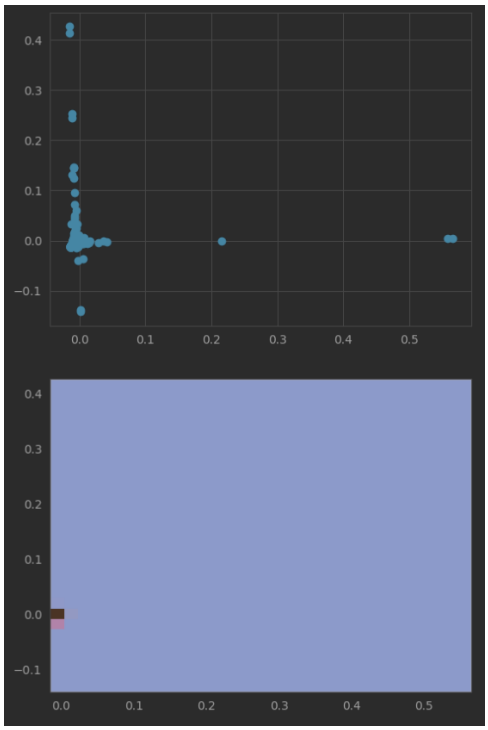
- presentation of the small degree stochastic walk tweak which improves diffusion map performance:

```
# Empirically this gamma seems to work well.
# It discourages taking edges to popular nhbs.
gamma = 0.9
```

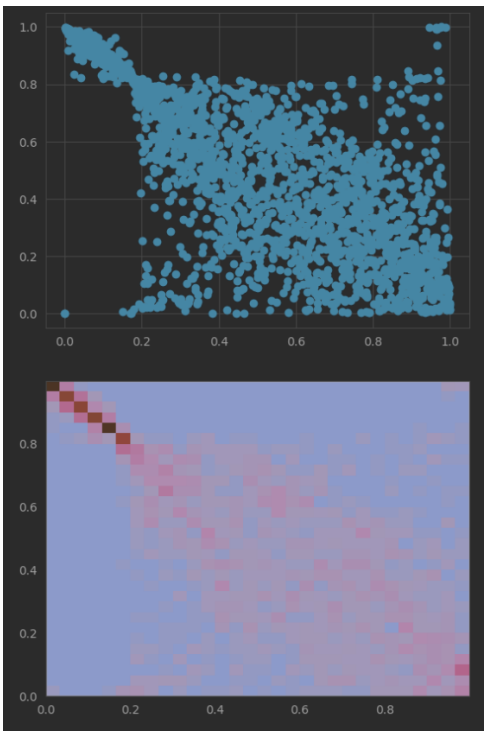


(a) 2d GIRG non-uniformed

(b) 2d GIRG uniformed



(c) socfb-Amherst41 non-uniformed



(d) socfb-Amherst41 uniformed

Figure 6.5: Diffusion map scatter plot of the first two extracted coordinates, with and without using an additional uniform square remapping

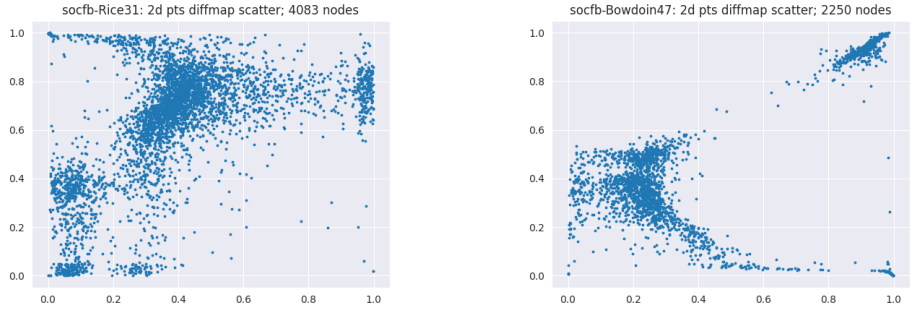


Figure 6.6: the diffmaps (here 2d, restricted and outer uniformified) don't have much point clustering - it's more higher diffusion level than caring so much about individual edges like in the converged case

```
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? Empricially seems to happen in 2D but not 1D?

Chapter 7

Likelihood Point Estimation

Contents

7.1 Introduction

Diffusion Maps in chapter 6 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].

7.2 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 $\theta^{j_1}, \theta^{j_2}, ...$ 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 $Q_u(\theta_u) = \prod_{v\neq u} p(e(u,v)|...)$.

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

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

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

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

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

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

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

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 Chung-Lu 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 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. 7.1 for some likelihood convergence curves on some example graphs.

7.2.2 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

(7.6)

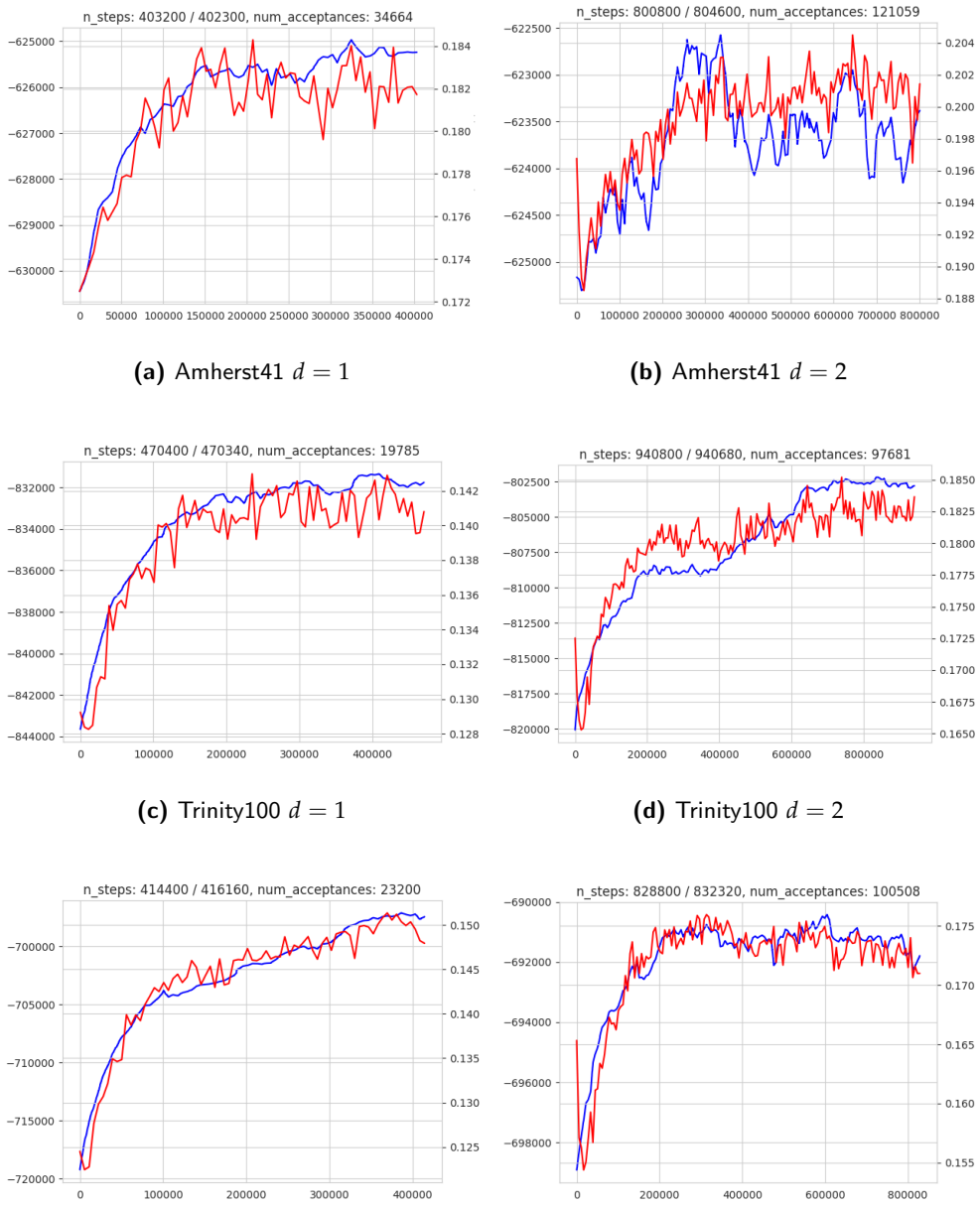
we can equivalently count $\frac{TP}{TP+FN}$ (Recall), the fraction of edges in the real graph that are successfully predicted, or $\frac{TP}{TP+FP}$ (Precision), the fraction of edges in the predicted graph that are also in the real graph, these numbers will be very similar. We call this metric "Percent 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.

7.3 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 $\boldsymbol{x}_u$ 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.



(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 7.1: 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

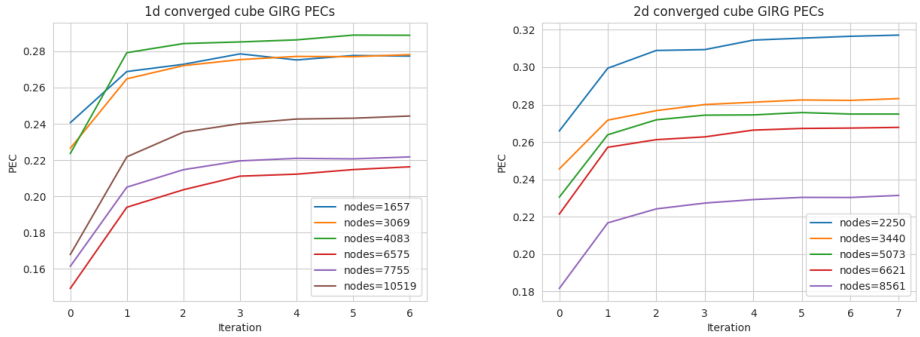


Figure 7.2: Direct ordered likelihood maximisation PEC convergence over iterations for some example socfb graphs.

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. 7.3 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.

TODO further improvements might be: updating weight estimates (adding into ordered MCMC, as per mercator). Making code run on GPU.

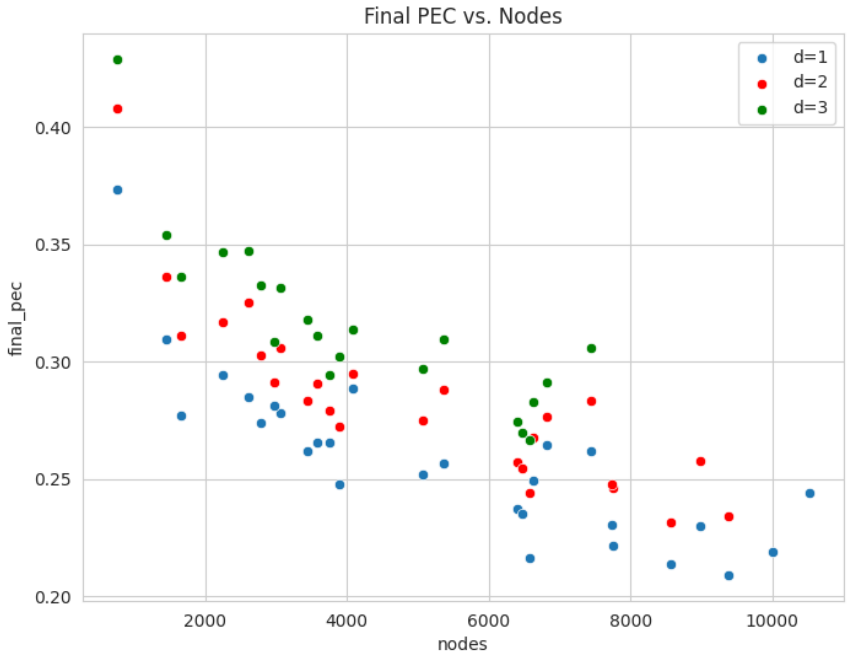
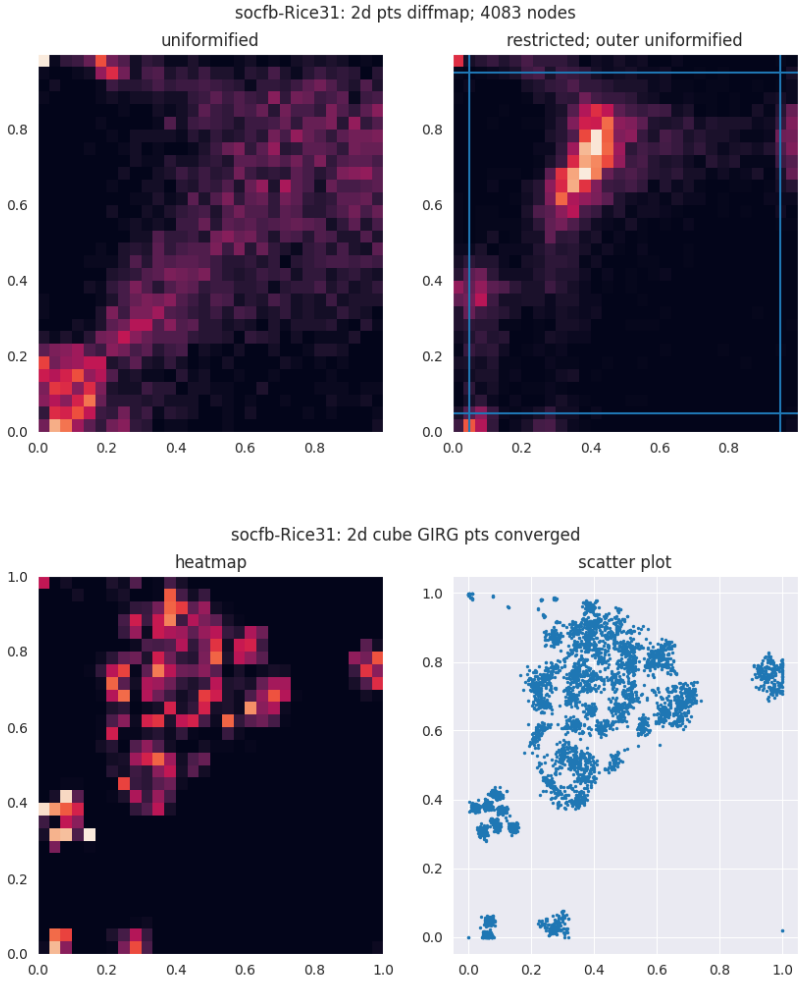
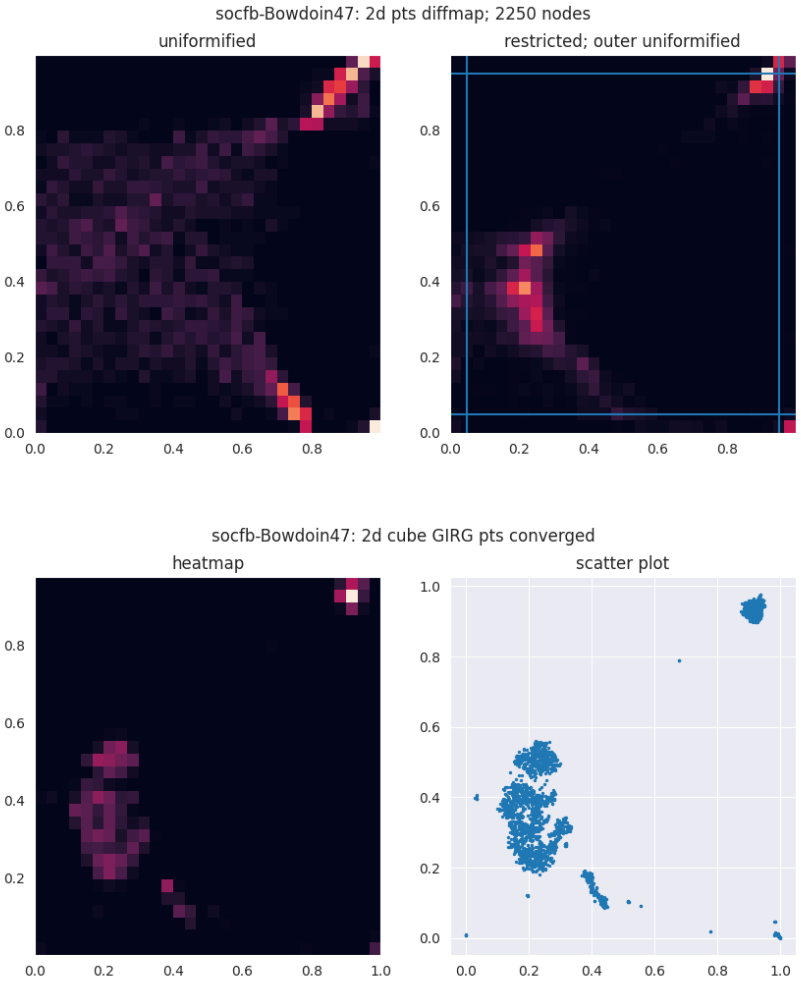
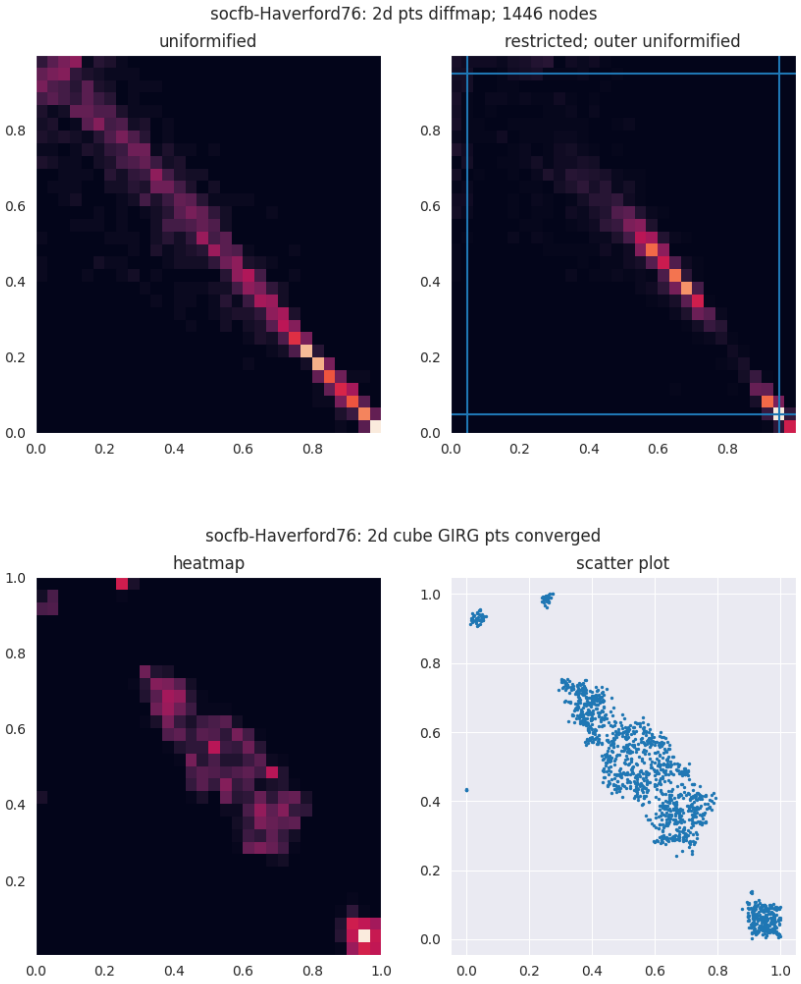


Figure 7.3: converged PECs for 1d, 2d, 3d cube GIRGs. Missing some 3d numbers as batch job exceeded allotted time.

Figure 7.4: 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-Bowdoin47: 2d pts diffmap; 2250 nodes
uniformified
restricted; outer uniformified
socfb-Bowdoin47: 2d cube GIRG pts converged
heatmap
scatter plot


socfb-Haverford76: 2d pts diffmap; 1446 nodes
uniformified
restricted; outer uniformified
socfb-Haverford76: 2d cube GIRG pts converged
heatmap
scatter plot

Chapter 8

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.

8.1 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. $M_1 = \mathscr{G}_{1D\,\mathrm{GIRG}}$ 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 $M_{d=1}, M_{d=2}, M_{d=3}, \ldots$ set of models for different dimensional GIRGs with some kind of decaying $p(1d\ \mathrm{GIRG}) > p(2d\ \mathrm{GIRG}) > p(3d\ \mathrm{GIRG}) > \ldots$.

For now focussing on just M_1 vs M_2,

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

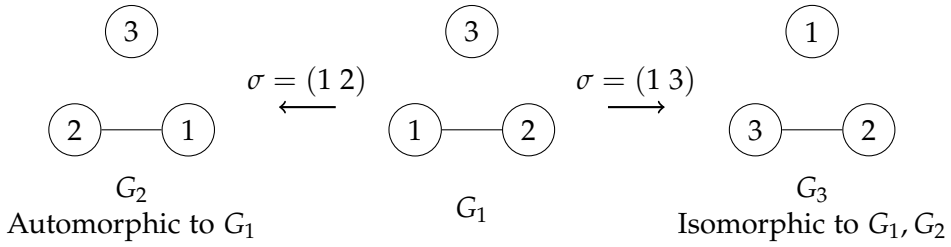


Figure 8.1: Example automorphic/isomorphic graphs

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 $p(D|M_1), p(D|M_2)$.

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 $\theta = c, \{\boldsymbol{x}_u\}_{u\in V}$, then we could Monte Carlo sample $\theta \sim p(\theta)$ from the prior to get an estimate for

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

The simplified notation is dropping the $|M_1$ as we 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. 8.1. 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*

($G \xrightarrow{\sigma} G'$), 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 $\theta = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ of the three nodes. The Chung-Lu has identical $p(G')$ for each of the 6 ismorphisms. The GIRG model does not - if $\boldsymbol{x}_1, \boldsymbol{x}_2$ are close, and $\boldsymbol{x}_3$ 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 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 $\mu(G) = E_{G' \sim \mathscr{G}_{\text{d-GIRG}}}[k(G, G')]$.

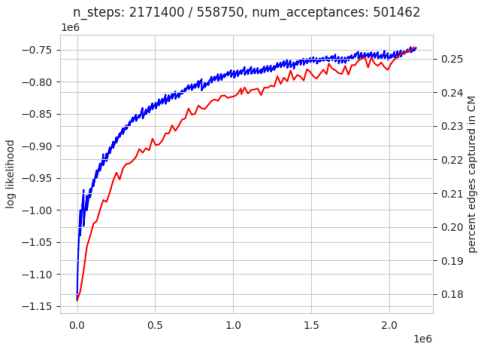
We see in fig. 8.2 that Bayes Factor model comparison wise, 1D GIRGs are superior, even though according to edge accuracy, 2D GIRGs achieve a slight edge.

8.2 Graph Kernel Introduction

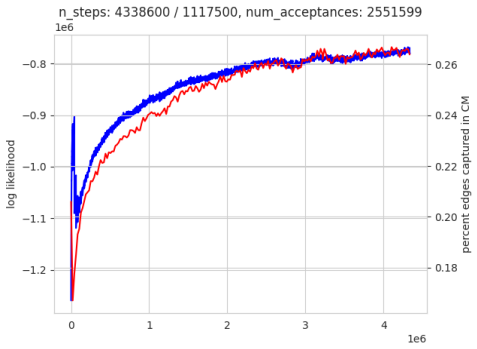
Graph Kernels provide a means for a similarity metric between graphs. They're ideally a positive semidefinite function $k : \Gamma \times \Gamma \to \mathbb{R}$, where Γ is the set of all graphs. Such a function exists if and only if there is a corresponding feature map resentation of $\phi : \Gamma \to \mathcal{H}_\phi$ 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 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!

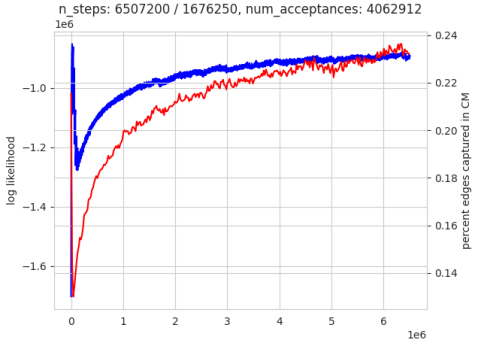
Another benefit of graph kernels is, as a similarity metric, they give easier direct comparison between graphs. In the previous sections' proposed paradigm, we can directly compare the similarity of a real graph G with two synthetic graphs G_1, G_2, produced from different models M_1, M_2, using $k(G,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



(a) $d = 1$



(b) $d = 2$



(c) $d = 3$

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

8.3 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.

8.3.1 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. 8.4.

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

8.3.3 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. 8.3 and fig. 8.4.

E.g. in fig. 8.4, the random walk kernel faield the basic test of having $k(G \sim \mathscr{G}_1, G_1' \sim \mathscr{G}_1) > k(G \sim \mathscr{G}_1, G_2' \sim \mathscr{G}_2)$, where $\mathscr{G}_1$ is a 3D copy weight cube GIRG, and $\mathscr{G}_2$ is copy weight Chung-Lu.

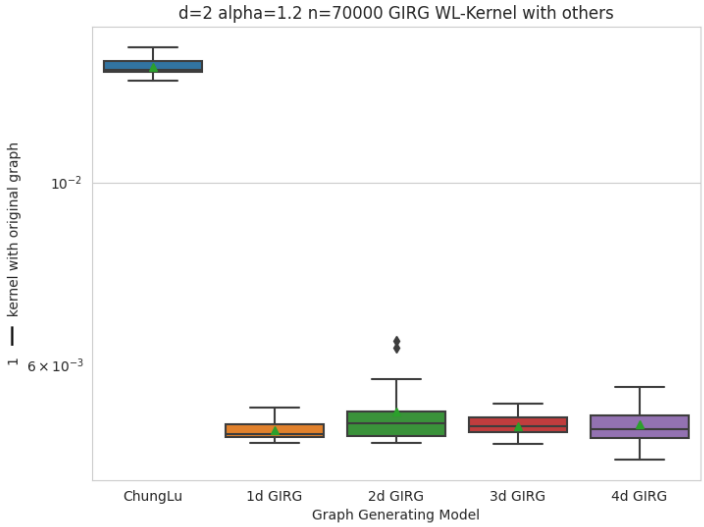


Figure 8.3: WL-Kernel of a d=2, alpha=1.2, n=70000 Torus GIRG with other generated graphs (13 per model). All the GIRGs are more similar to the original than Chung-Lu, but we cannot differentiate between GIRGs of different dimensions.

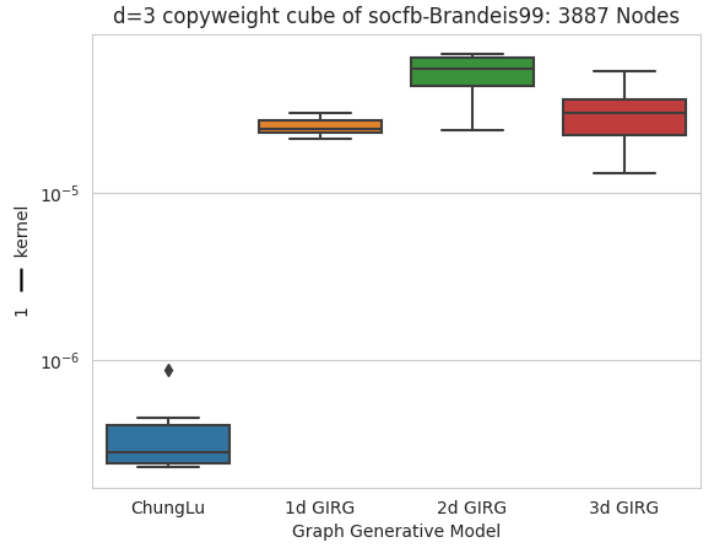


Figure 8.4: RW-Kernel of a d=3 copy weight cube GIRG fit to socfb-Brandeis99 (matching number of edges and local clustering coefficient), with other generated graphs (6 per model type). Chung-Lu graphs have highest similarity to the original, despite it being a 3D GIRG

Chapter 9

Conclusion

Contents

9.1 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.

[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.

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

[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.

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.