



โครงการ

การเรียนการสอนเพื่อเสริมประสบการณ์

ชื่อโครงการ การเกิดรูปทรงเรขาคณิตแบบสุ่มด้วยวิธีทางกลศาสตร์เชิงสถิติ
Statistical Mechanics of Emergent Geometry in Random Graphs

ชื่อนิสิต นายภาวัต อัครพิพัฒนา เลขประจำตัว 6033428123

ภาควิชา ฟิสิกส์

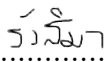
ปีการศึกษา 2563


คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย


Project Title Statistical Mechanics of Emergent Geometry in Random Graphs
By Pawat Akarapipattana
Field of Study Physics
Project Advisor Thiparat Chotibut, Ph.D.
Academic Year 2020


This report is submitted to the Department of Physics, Faculty of Science, Chulalongkorn University, in partial fulfillment of the requirements for the degree of Bachelor of Science.

This report has been approved by the committee:

.......... Chairwoman
(Rangsim Chanphana, Ph.D.)

.......... Committee
(Asst. Prof. Pavin Ittismai, Ph.D.)

.......... Project Advisor
(Thiparat Chotibut, Ph.D.)

.......... Project Co-advisor
(Oleg Evnin, Ph.D.)

STATISTICAL MECHANICS OF EMERGENT GEOMETRY IN RANDOM GRAPHS

by

Pawat Akara-pipattana

A thesis submitted in conformity with the requirements
for the degree of Bachelor of Science

Department of Physics, Faculty of Science
Chulalongkorn University

© Copyright 2021 by Pawat Akara-pipattana

Statistical Mechanics of Emergent Geometry in Random Graphs

Pawat Akara-pipattana
Bachelor of Science

Department of Physics, Faculty of Science
Chulalongkorn University
2021

Abstract

How spacetime emerges from featureless nothingness is one of the most intriguing questions in fundamental physics. In this thesis, we take on the random geometry approach to study discretized spacetime and follow the assumption that, in the simplest form, geometric structures may arise from random connections between dots under certain rules. We study a family of random graph models called Exponential Random Graph Models (ERGMs). Although this family was extensively investigated in the network science community as a proxy to study real-world social networks, its strength is in its formulation as a Gibbs-Boltzmann distribution in equilibrium statistical mechanics. Thus, one can modify, analyze, and simulate the ensemble using familiar tools from statistical mechanics. We are interested in modifying the basic ERGMs to arrive at a random graph model that possesses emergent geometric properties. This would serve as a proof-of-principle that geometric spacetime may emerge from randomly connected dots.

Our study leads to novel classes of random graphs whose edges can self-assemble themselves into both simple geometric primitives (e.g. triangles) and more complex structures (e.g. hexagons). The number of such structures is relatively large compared to the amount of dots and connections available in the graph. Lastly, but interestingly, our model is free from the graph collapse problem that is often observed in the traditional ERGMs.

Acknowledgments

First of all, I would like to express my gratitude toward my advisor, Dr. Thiparat Chotibut, who introduces me to the world of complex systems and statistical physics, and equips me with the knowledge necessary to pursue research in this field.

I owe a deep sense of gratitude to Dr. Oleg Evnin for spending his precious time guiding and pushing me through this project. I learned a lot not only on the relating topics but also on academic thinking and research methodology during the time I work with him. Without him, I alone would not be able to complete this work.

I would like to also thank Extreme Condition Physics Research Laboratory and Chula Intelligent and Complex Systems of Chulalongkorn University for providing computational resources crucial to this project. This project would have been delayed by months had it not had this resource. I also thank Dr. Teerachote Pakornchote and Krittin Phornsiricharoenphant, the administrator of high-performance computing clusters of these two research groups, for their effort of maintaining the clusters throughout this project.

I would like to thank Gephi team who develops and maintains network visualization software Gephi [2] which is used to generate all graph visualization in this thesis.

Last but not least, I would like to thank Phanwasa Sapprasert as well as my friends and family for staying by my side and providing mental supports during the duration of my study and my project.

Contents

1	Introduction	1
1.1	Undirected Graph	2
1.2	Exponential Random Graph Models	2
1.3	Erdős–Rényi Graph	3
1.4	Two-star Model	4
1.4.1	Dense Regime	5
1.4.2	Sparse Regime	10
1.4.3	Graph Collapse Problem (Degeneracy Problem)	14
2	Geometric Graph Models	15
2.1	The Modified Square Model	15
2.2	The Modified Triangle Model	16
3	Numerical Results	18
3.1	The Modified Triangle Model	18
3.2	The Modified Square Model	22
4	Conclusion	25
A	Numerical Methods	26
A.1	Markov Chain Monte Carlo	26
A.1.1	Markov Chain	26
A.1.2	Detailed Balance	27
A.1.3	Ergodicity	28
A.1.4	Acceptance Ratios	28
A.1.5	The Metropolis Algorithm	28
A.2	Optimization of The Modified Model	29
B	Code	31
B.1	The Erdős–Rényi Graph	31

B.2	The Two-star Model	31
B.2.1	Initialization	32
B.2.2	Markov Process	32
B.2.3	Full Code	33
B.3	The Modified Two-star Model	38
	Bibliography	42

List of Figures

- 1.1 Connectance plot as a function of J with $\alpha = -J$. MF denotes mean-field solution and MCMC denotes the result of my simulation. The bifurcation indicate a phase transition at $J = 1$. The graphs (a), (b), and (c) in the second row are quantitative visualizations of two-star graphs at $J = 0.5$, $J = 1.5$ with high mean degree initial state, and $J = 1.5$ with low mean degree initial state. The symmetry around connectance = 0.5 is due to the edge-hole symmetry in the two-star Hamiltonian. In the graph visualizations, edges overlap with each other creating a visual opacity in the graph; the level of opacity can be roughly thought of as a density of edges of the corresponding vertex. 11

- 1.2 Variance of node degree as a function of J along the line $\alpha = -J$ at $N = 300$. The sharp peak at $J = 1$ indicates a phase transition. We observed that the fluctuations (reflected in the error bar) is larger in the region $J < 1$ than when $J > 1$. 12

- 1.3 Plot of log connectivity ($\log \langle k \rangle / \log N$) and log star-density ($\log (\langle k^2 \rangle - \langle k \rangle) / \log N$) as a function of β for $\alpha = -0.5$, and $4 - \frac{1}{2} \log N$ with $N = 300$. MCMC denotes the result from the simulation, MF denotes mean-field solution, and FC denotes finite connectivity solution from Annibale's work (1.41), (1.42). On the last row, we have a visualization of a graph at $\alpha = -0.5 - \frac{1}{2} \log 300$, $\alpha = 4 - \frac{1}{2} \log 300$ to illustrate typical configurations in this sparse regime. Nodes with degree 0 are not shown in the visualization. 13

- 1.4 A two-star graph at $\alpha = -2.852$, $\beta = -10$, and $N = 300$. This graph with mean degree 6 and degree variance exactly 0 will be used as a baseline for non-geometric graph to be discussed further in Chapter 3. 14

3.1	Typical graph configurations sampled from the modified triangle model plotted over the parameter space β and σ . α is tuned so that the mean degree is close to 6 (the range of mean degree in this plot is ± 0.1). The two percentages under each graph are the percentage of triangles and hexagons respectively.	19
3.2	Convergence of the mean degree, degree variance, and the percentage of triangles, and of hexagons from the modified triangle model at $\alpha = 177$, $\beta = -20$, and $\sigma = 100$. Each quantity is re-scaled by its equilibrium value so that they are all normalized to 1. The horizontal axis is the Monte Carlo step count in units of 10 million steps.	20
3.3	The mean degree, degree variance, and the percentage of triangles and of hexagons of the modified triangle model from Monte Carlo simulation at $N = 2000$, $\alpha = 177$, $\beta = -20$, $\sigma = 100$. The horizontal axes is the density of Erdos-Renyi graph initial state. The green crosses are mean values for each initial condition averaged over 10 Monte Carlo runs.	21
3.4	Diameter values of graphs in the modified triangle model plotted as a function of graph size N . At each size, 50 samples are simulated. The size orange dots represents the frequencies of data at each value of the diameter. The blue line is the best fit line, which gives the scaling law of the diameter $N^{0.1845 \pm 0.0083}$. This implies that the structures of graphs from the modified triangle model are more similar to lattice than Erdős–Rényi graph.	22
3.5	A very large modified triangle graph obtained from the simulation at $N = 5000$, $\alpha = 177$, $\beta = -20$, $\sigma = 100$	23
3.6	The mean degree, degree variance, and the percentage of squares and of cubes of the modified square model from Monte Carlo simulation at $N = 2000$, $\alpha = 80$, $\beta = -20$, $\sigma = 100$. The horizontal axes is the density of Erdos-Renyi graph initial state. The green crosses are mean values for each initial condition averaged over 10 Monte Carlo runs.	24

Chapter 1

Introduction

The origin of spacetime is arguably one of the most important and intriguing questions of fundamental physics. One approach that allows for the intuitive interpretation of emergent spacetime as emergent discretized geometry is through the statistical mechanics formulation of random graphs. A pioneering work in this direction was put forward by Trugenberger and Kelly; they constructed a random graph model whose action (Hamiltonian) is based on Ollivier curvature, an analog of Ricci curvature defined on a graph [3, 4, 10]. However, their models express many unnatural constraints which raises the question whether one can construct a simpler model that still leads to desirable emergent geometry. Our goal is to seek a minimal prescription of random graph ensembles, such that macroscopic numbers of geometric structures can emerge naturally in our ensembles. Without resorting to controversial assumptions of Quantum Gravity, we will focus on emergent discretized geometry within the self-contained context of random graph models.

The organization of the thesis is as follows. In this Introduction Chapter, we shall begin by introducing random graph models. Then we will introduce and review important properties of the two-star model, one of the simplest classes of Exponential Random Graph Models (ERGMs) on which our models are built upon. In Chapter 2, we introduce the modifications to the original two-star model and show that these modifications are able to meet our expectations. We then report these desirable properties arising from our models extensively in Chapter 3. Finally, we provide a brief comprehensive guide to the Monte Carlo algorithm used in our simulations in Appendix A.

1.1 Undirected Graph

In this section we introduce background concepts and terminologies necessary to understand random graph models. We'll focus on an undirected graph without a self-loop. The undirected graph G is defined on a set of nodes V and a set of edges $E \subseteq \{\{x, y\} | x, y \in V \text{ and } x \neq y\}$. This abstract definition of graph is hard to work with, so we will instead work with a concrete matrix representation of a graph called the *adjacency matrix* \mathbf{A} . A graph with N nodes is represented by an $N \times N$ adjacency matrix. The ij component of the adjacency matrix encodes the connection of the graph via the following definition:

$$A_{ij} = \begin{cases} 0 & \text{if } i \text{ is not connected to } j \\ 1 & \text{if } i \text{ is connected to } j \end{cases}. \quad (1.1)$$

With this, we can define the total connection of a node i known as *degree*

$$k_i = \sum_j A_{ij}. \quad (1.2)$$

A *diameter* of a graph is defined to be the longest shortest path between each pair of nodes. The diameter will later play a crucial role in determining large scale structures of a graph.

In the context of the random graph model, one may encounter the term *motif*. Basically, motifs are subgraphs of interest that we encode in the Hamiltonian, which shall be made clear in later chapters.

1.2 Exponential Random Graph Models

Random graph models have been extensively investigated by sociologists to study the structures of real-world networks. We shall focus on a family of models known as *Exponential Random Graph Models* (ERGMs). The ERGM is the least biased way to model the probability of finding a graph in a collection of all possible graphs, formally known as *graph ensemble*, without encoding *a priori* knowledge of the graph structure except for the given measurable (observable) quantities such as mean degree, degree variance, and etc. Such most noncommittal model can be realized by enforcing the probability distribution of a graph in a graph ensemble to maximize the Shannon's entropy subjected to the constraints from the observables. This can be done using the Lagrange multipliers with the following Lagrange function \mathcal{L} , to be maximized

with respect to $P(G)$,

$$\mathcal{L} = \sum_G P(G) \log P(G) + \lambda \left(\sum_G P(G) - 1 \right) + \sum_i \theta_i \left(\sum_G g_i(G) P(G) - g_i^* \right), \quad (1.3)$$

where the Lagrange multiplier λ constrains $P(G)$ to be a probability distribution and must be normalized to 1, and the Lagrange multiplier θ_i enforces that the ensemble average g_i from the probabilistic model must reproduce the empirical average (observable) g_i^* . For example, to encode the mean degree into the set of measurable constraints, g_i would be $\sum_j k_j$ and g_i^* would be the empirical mean degree obtained from networks data of interests.¹ Recalling that a graph G can be represented concretely as its adjacency matrix configurations $\mathbf{A} \in \{0, 1\}^{N \times N}$, then solving the optimization problem (1.3) gives the probability of finding a graph with a specific configuration to be

$$P(G) = \frac{1}{Z} e^{-H_{\boldsymbol{\theta}}(\mathbf{A})}, \quad (1.4)$$

where $Z = \sum_{\{\mathbf{A}\}} e^{-H_{\boldsymbol{\theta}}(\mathbf{A})}$ is the partition function and $H_{\boldsymbol{\theta}}(\mathbf{A}) = -\sum_i \theta_i g_i(\mathbf{A})$ is called the *Hamiltonian* of the graph. We shall call the entire collection of possible graphs with its associated probability distribution a *graph ensemble*. Note that we end up with a Boltzmann distribution of a graph ensemble, hence the name ERGMs, with the temperature factor $k_B T = 1$. The reformulation of random graphs as the equilibrium statistical mechanics ensemble allows us to modify the Hamiltonian fairly easily without losing the explainability of the probabilistic models. Namely, one can add desirable or undesirable motifs to the graph Hamiltonian to study a graph ensemble of interest, which we will later discuss.

1.3 Erdős–Rényi Graph

The simplest model in the ERGM family is the Erdős–Rényi graph. In this section, we shall briefly review this model since it serves as the simplest random graph model.

Erdős–Rényi graph is defined as a random graph ensemble in which the number of vertices is fixed and each edge has a fixed probability of being present or absent, independently of the other edges [6]. Traditionally, the probability of finding a graph $G_{N,m}$ in the Erdős–Rényi ensemble with fixed N vertices, fixed m edges with the

¹Determining probabilistic model's parameters from observable statistics is the essence of inverse modeling (statistical inference), as opposed to the forward modeling case where one would predetermine the model parameters and study the behaviors of the model (deductive logic).

probability of any edge to be present $p \equiv \frac{m}{N}$ is

$$P(G_{N,m}) = p^m (1-p)^{\binom{N}{2}-m}. \quad (1.5)$$

We can rewrite the probability of the Erdős–Rényi graph in the language of ERGM in exponential form as

$$\begin{aligned} P(G_{N,m}) &= \exp [m \log p + ((\binom{N}{2}) - m) \log (1 - p)] \\ &= \exp \left[m \log \left(\frac{p}{1-p} \right) + (\binom{N}{2}) \log (1 - p) \right]. \end{aligned} \quad (1.6)$$

With a proper normalization and upon the substitution $m \equiv \sum_i k_i$, Eqn. (1.6) translates into

$$P(G) = \frac{1}{Z} e^{\alpha(p) \sum_i k_i}, \quad (1.7)$$

where $\alpha(p) = \log \left(\frac{p}{1-p} \right)$. This ERGM form of the Erdős–Rényi graph also preserves the original definition; we can see that the probability depends explicitly on the given (fixed) number of edges in the ensemble. Note that if we scale $\alpha(p)$ as $-\log N$, we get $p = 1/N$ and the degree is finite.

1.4 Two-star Model

In this section we review the simplest non-trivial model studied extensively in [8], the two-star model, to familiarize the readers with techniques and characters of slightly more complicated ERGMs. We shall focus on both analytical and numerical results to give the readers the impression of this model. For technical details of the numerical simulation and algorithm used please refer to Appendix A. In the two-star model, the Hamiltonian is proportional to two quantities: total degree and total number of two-stars. A two-star of a node i is defined as the number of path of length two passing through the node i . For the node i with degree k_i , the two-star can thus be expressed as $k_i(k_i - 1)/2$. We can then write the Hamiltonian as

$$H(\mathbf{A}) = -\frac{\theta_1}{2} \sum_i k_i(\mathbf{A}) - \frac{\theta_2}{2} \sum_i k_i(\mathbf{A}) (k_i(\mathbf{A}) - 1). \quad (1.8)$$

The factor $1/2$ arises from the fact that the sum of degree over all nodes double-count the edges. This Hamiltonian can be simplified further to

$$H(\mathbf{A}) = -\alpha \sum_i k_i(\mathbf{A}) - \beta \sum_i k_i^2(\mathbf{A}) \quad (1.9)$$

where $\alpha = (\theta_1 - \theta_2)/2$ and $\beta = \theta_2/2$ are ensemble parameters to be specified.

1.4.1 Dense Regime

Mean-field Solution

The dense regime in the two-star model had been studied analytically using mean-field approximation [8]. The key idea of this method is to approximate all of the interactions by an effective field with the value of the mean of all interactions. This works well in the regime where the fluctuations is suppressed and the mean is a good representative of overall interaction, such as in the dense regime here where interactions are approximately all-to-all.

We begin with the Hamiltonian (1.8) in term of the adjacency matrix \mathbf{A}

$$\begin{aligned}
H(\mathbf{A}) &= -\frac{\theta_1}{2} \sum_i \sum_{j \neq i} A_{ij} - \frac{\theta_2}{2} \sum_i \sum_{j \neq i} \sum_{k \neq i, j} A_{ij} A_{jk} \\
&= -\frac{1}{2} \sum_i \sum_j \left[A_{ij} \left(\theta_1 + \theta_2 \sum_{k \neq i, j} A_{jk} \right) \right] \\
&= -\sum_i \sum_{j > i} A_{ij} \left((\theta_1 - \theta_2) + \theta_2 \sum_k A_{jk} \right) \\
&= -\sum_{i < j} A_{ij} \left(2\alpha + 2\beta \sum_k A_{jk} \right).
\end{aligned} \tag{1.10}$$

To linearize the quadratic interaction in the adjacency matrix, we can adopt the mean field approximation by replacing A_{jk} with its ensemble average $\langle A_{jk} \rangle = (N-1)p$ where p is the probability to an edge to be present. We thus have

$$H(\mathbf{A}) \approx -\lambda(p) \sum_{i < j} A_{ij}, \tag{1.11}$$

where

$$\lambda(p) = 2\alpha + 2\beta(N-1)p \approx 2\alpha + 2\beta Np. \tag{1.12}$$

Now the partition function can be factorized as the product of the partition function of the non-interacting systems

$$Z = \sum_A e^{\lambda(p) \sum_{i < j} A_{ij}} = \prod_{i < j} \sum_A e^{\lambda(p) A_{ij}} = \prod_{i < j} (1 + e^{\lambda(p)}) = (1 + e^{\lambda(p)})^{N(N-1)/2}. \tag{1.13}$$

Recalling that $k_B T$ is set to 1, then the *free energy* is given by

$$F = -\log Z = -\frac{N(N-1)}{2} \log(1 + e^{\lambda(p)}). \quad (1.14)$$

Recall also that we can derive the expectation value of the observables from the free energy

$$\sum_{i<j} \langle A_{ij} \rangle = -\frac{\partial F}{\partial \lambda} = \frac{N(N-1)}{2} \frac{e^{\lambda(p)}}{1 + e^{\lambda(p)}}. \quad (1.15)$$

On the other hand, the expected number of edges in the network where each edge is randomly drawn with probability p can be calculated directly from

$$\sum_{i<j} \langle A_{ij} \rangle = \frac{N(N-1)}{2} p. \quad (1.16)$$

Equating (1.15) and (1.16) gives the following self-consistent equation for p

$$p = \frac{e^{\lambda(p)}}{1 + e^{\lambda(p)}} = \frac{1}{2} [1 + \tanh(\alpha + \beta N p)], \quad (1.17)$$

where we use the identity $e^x/(1 + e^x) = [1 + \tanh(x/2)]/2$ in the second equality. In the dense regime, it is convenient to scale $\beta N \rightarrow 2J$ so that the two-star term in the Hamiltonian $\beta \sum_i k_i^2$ is in the same order $\mathcal{O}(N)$ as the degree term. The self-consistent equation then becomes

$$p = \frac{1}{2} [1 + \tanh(2Jp + \alpha)]. \quad (1.18)$$

Park and Newman show that (1.18) has one unique solution for $J < 1$ and three solutions with two stable solutions for $J > 1$ and α sufficiently close to $-J$. This leads to the bifurcation at $J_c = 1$ corresponding to a *continuous* phase transition to a symmetry broken state with two phases: a high density phase and a low density phase. The condition $\alpha \approx -J$ is important because along this parametrization, the Hamiltonian has an edge-hole symmetry, which means the Hamiltonian is symmetric under the substitution of k_i by $N - k_i$, and thus can display two phases. In fact, for any $J > 1$, there is a first-order phase transition between a high and a low density phase. The two are separated by the critical line $\alpha = -J$ in the parameter space [8]. We display the bifurcation diagram in Fig.1.1.

Saddle Point Approximation

The mean-field approximation can only take us so far as to the regime where fluctuations is negligible. To see the first-order fluctuations correction to the mean-field

solution, we'll employ the saddle point approximation. To use the saddle point approximation, we first transform the Hamiltonian using the Hubbard-Stratonovich transformation into the set of auxiliary variable $\{p_i\}$ with the identity

$$\exp(\beta k_i^2) = \sqrt{\frac{\beta}{\pi}} \int_{-\infty}^{\infty} dp_i \exp(-\beta p_i^2 + 2k_i p_i \beta). \quad (1.19)$$

So that the partition function becomes

$$\begin{aligned} Z &= \sum_{\{\mathbf{A}\}} e^{-H(\mathbf{A})} \\ &= \left[\frac{\beta}{\pi}\right]^{N/2} \int \mathcal{D}\mathbf{p} \exp\left(-\beta \sum_i p_i^2\right) \times \sum_{\{\mathbf{A}\}} \exp\left(\sum_i 2(\beta p_i + \alpha)k_i\right), \end{aligned} \quad (1.20)$$

where $\int \mathcal{D}\mathbf{p}$ indicates that the integral is performed over all field $\{p_i\}$. Here, we interpret p_i as an auxiliary field as usual, but as we'll see shortly, it can be interpreted as the probability for the edge to be present as well.

Recalling that $k_i = \sum_j A_{ij}$ and that \mathbf{A} is symmetric, we can rewrite the exponent of the last term as

$$\begin{aligned} \sum_i (2\beta p_i + \alpha)k_i &= \sum_{ij} (2\beta p_i + \alpha)A_{ij} \\ &= \sum_{i>j} (2\beta(p_i + p_j) + 2\alpha)A_{ij}. \end{aligned} \quad (1.21)$$

Then we can evaluate the summation over ensemble as

$$\begin{aligned} \sum_{\{\mathbf{A}\}} \exp\left(\sum_i 2(\beta p_i + \alpha)k_i\right) &= \prod_{i>j} \sum_{A_{ij}=0,1} \exp[(2\beta(p_i + p_j) + 2\alpha)A_{ij}] \\ &= \prod_{i>j} (1 + \exp[2\beta(p_i + p_j) + 2\alpha]) \end{aligned} \quad (1.22)$$

Now, we can write the partition function in the form

$$Z = \int \mathcal{D}\mathbf{p} e^{-\mathcal{H}(\mathbf{p})}, \quad (1.23)$$

with the new Hamiltonian

$$\mathcal{H}(\mathbf{p}) = \beta \sum_i p_i^2 - \frac{1}{2} \sum_{i \neq j} \log(1 + e^{2\beta(p_i + p_j) + 2\alpha}) - \frac{N}{2} \log\left(\frac{\beta}{\pi}\right). \quad (1.24)$$

Now that we have transformed a discrete combinatorial counting into a continuous

scalar field counting in the partition function, we can take $N \rightarrow \infty$ limit and proceed with the saddle point approximation. First, let's reproduce the mean-field solution by looking for the solution that extremizes the Hamiltonian;

$$\frac{\partial \mathcal{H}}{\partial p_i} = 0 = 2\beta p_i - \beta \sum_{i(\neq j)} (\tanh(\beta(p_i + p_j) + \alpha) + 1). \quad (1.25)$$

The solution of this equation is

$$p_0 = \frac{1}{2} [\tanh(2\beta N p_0 + \alpha) + 1], \quad (1.26)$$

where we substitute p_i and p_j with p_0 for every i, j . Note that this method gives a similar mean-field solution as the one in the previous section.

Note also that from the partition function, we can calculate the expected degree

$$\langle k \rangle = \frac{1}{N} \sum_i \langle k_i \rangle = -\frac{1}{N} \frac{\partial \log Z}{\partial \alpha} = \frac{1}{2N} \sum_{i \neq j} \langle \tanh(\beta N(p_i + p_j) + \alpha) + 1 \rangle = (N-1)p_i, \quad (1.27)$$

and see that p_i behaves like the probability for the edge to be present as in the previous section, hence the same notation we adopted for the auxiliary variables.

To evaluate the first-order fluctuations correction to the mean-field solution, we expand the Hamiltonian to second order in \mathbf{p}

$$\mathcal{H}(\mathbf{p}) = \mathcal{H}(\mathbf{p}_0) + d\mathbf{p}^\top \mathbf{M} d\mathbf{p} + \mathcal{O}(d\mathbf{p}^3), \quad (1.28)$$

where $d\mathbf{p} \equiv \mathbf{p} - \mathbf{p}_0$ and \mathbf{M} is Hessian matrix of \mathcal{H} with respect to \mathbf{p} , evaluated at \mathbf{p}_0 . We then change the variables to $\boldsymbol{\xi} \equiv \mathbf{Q}^{-1} d\mathbf{p}$, where \mathbf{Q} is the eigenvector matrix of \mathbf{M} . Now, we can write $d\mathbf{p}^\top \mathbf{M} d\mathbf{p} = d\mathbf{p}^\top \mathbf{Q} \mathbf{Q}^{-1} \mathbf{M} \mathbf{Q} \mathbf{Q}^{-1} d\mathbf{p} = \boldsymbol{\xi}^\top \mathbf{Q}^{-1} \mathbf{M} \mathbf{Q} \boldsymbol{\xi}$ so \mathbf{M} is diagonalized under this basis transformation and, after neglecting $\mathcal{O}(d\mathbf{p}^3)$, so we can write

$$\mathcal{H}(\mathbf{p}) \approx \mathcal{H}(\mathbf{p}_0) + \sum_i \lambda_i \xi_i^2 \quad (1.29)$$

with λ_i being the corresponding eigenvalue of the i^{th} column of \mathbf{M} . Notice that the Jacobian for the change of variable to $\boldsymbol{\xi}$ coordinate is 1 (Orthogonal transformation), we can write the partition function as

$$Z \approx \int \mathcal{D}\boldsymbol{\xi} e^{-\mathcal{H}(\mathbf{p}_0) - \sum_i \lambda_i \xi_i^2} = \underbrace{\left(\frac{1}{\sqrt{\det(\mathbf{M})}} \right)}_{\text{fluctuations correction}} \underbrace{e^{-\mathcal{H}(\mathbf{p}_0)}}_{\text{mean-field}}. \quad (1.30)$$

The elements of \mathbf{M} are

$$M_{ij} = \begin{cases} -4J^2 p_0(1-p_0) & \text{for } i \neq j \\ (N-1)(2J-4J^2 p_0(1-p_0)) & \text{for } i = j \end{cases} \quad (1.31)$$

where we write J instead of βN . This gives the determinant of \mathbf{M} to be

$$\det(\mathbf{M}) = (2(N-1)J)^N (1-2Jp_0(1-p_0))^{N-1} (1-4Jp_0(1-p_0)). \quad (1.32)$$

We can now calculate the mean degree $\langle k \rangle$ and the mean squared degree $\langle k^2 \rangle$ from the free energy from the new Hamiltonian to give

$$\langle k \rangle = Np + \frac{2Jp_0(1-p_0)(1-2p_0)}{(1-4Jp_0(1-p_0))(1-2Jp_0(1-p_0))}, \quad (1.33)$$

$$\langle k^2 \rangle = N^2 p_0^2 + \frac{Np_0(1-p_0)(1-4Jp_0^2)}{(1-4Jp_0(1-p_0))(1-2Jp_0(1-p_0))}, \quad (1.34)$$

where we use the approximation $N-1 \approx N$ which holds true for $N \gg 1$. The leading order term in N are the same as the mean-field solution. The second terms are the result of Gaussian fluctuation around mean-field which vanishes in the limit of large N . One can compute the degree variance from

$$\langle k^2 \rangle - \langle k \rangle^2 = N \frac{p_0(1-p_0)}{1-2Jp_0(1-p_0)}, \quad (1.35)$$

and might naively expect it to diverge at the second-order phase transition point. However, degree variance plot in Fig.1.2 doesn't show divergence; the variance is continuous at the transition point. In fact, the quantity that diverges is the variance of the number of edges in the network since it turns out to be the susceptibility of the two-star model. The susceptibility of *total degree* defined as $m \equiv \sum_i k_i$ is the second derivative of the free energy F with respect to α :

$$\langle m^2 \rangle - \langle m \rangle^2 = \frac{\partial^2 F}{\partial \alpha^2} = N \frac{2p_0(1-p_0)}{1-4Jp_0(1-p_0)}. \quad (1.36)$$

I successfully reproduce the analytical and numerical results from the work of Park and Newman. In their work, they plot the *connectance* defined as $\langle k \rangle / (N-1)$ versus the degree variance as a function of $J = \beta(N-1)$ setting $\alpha = -J$ and $N = 1000$. Their result shows a bifurcation in connectance plot and a sharp peak in degree variance at $J = 1$ indicating a second-order phase transition. I simulated graph ensemble with the same set of parameters but only used $N = 300$. Fig.1.2 shows that

the simulation agree with the mean-field solution nicely.

1.4.2 Sparse Regime

The motivation behind the exploration beyond the dense regime is that, in the dense regime, the mean connectivity is proportional to the number of nodes in the network which is rarely observed in real-world networks; e.g. increasing the population size does not imply one has more friends, on average, in a social network [1]. The more reasonable model for real-world networks would have its connectivity independent of the population size. Here we explore a more recent work by Annibale and Courtney [1] that studies the sparse regime of the two-star model.

In $N \gg 1$ limit, the leading order of the mean-field solution (1.33) is Np . If we want $\langle k \rangle$ to be $\mathcal{O}(N^0)$, we need to scale $\alpha \rightarrow \hat{\alpha} - \frac{1}{2} \log(N)$ and set $\beta = \mathcal{O}(1)$ so that

$$p = \frac{1}{2} \left[1 + \tanh \left(\beta N p + \hat{\alpha} - \frac{1}{2} \log N \right) + 1 \right] = \frac{\frac{1}{N} e^{2\beta N p + 2\hat{\alpha}}}{\frac{1}{N} e^{2\beta N p + 2\hat{\alpha}} + 1} \approx \frac{1}{N} e^{2\beta N p + 2\hat{\alpha}} \quad (1.37)$$

Setting $c = pN$, we get a new self-consistent equation for c

$$c = e^{2\beta c + 2\hat{\alpha}}, \quad (1.38)$$

which is independent of N . Now substitute c in (1.33) and (1.34) and we have the mean-field

$$\langle k \rangle = c \left(1 + \frac{\beta}{(1 - 2\beta c)(1 - \beta c)} \right), \quad (1.39)$$

$$\langle k^2 \rangle = c \left(c + \frac{1}{(1 - 2\beta c)(1 - \beta c)} \right). \quad (1.40)$$

However, the mean-field solution is not accurate in sparse regime since the fluctuations is now non-negligible. The disagreement between the mean-field prediction and the simulation is shown in Fig.1.3. Annibale and Courtney [1] solved this model in the limit $N \gg 1$ in the finite connectivity regime and it gives

$$\langle k \rangle = \frac{\sum_{k \geq 0} k g(k) e^{\alpha k + \beta k^2}}{\sum_{k \geq 0} g(k) e^{\alpha k + \beta k^2}}, \quad (1.41)$$

$$\langle k^2 \rangle = \frac{\sum_{k \geq 0} k^2 g(k) e^{\alpha k + \beta k^2}}{\sum_{k \geq 0} g(k) e^{\alpha k + \beta k^2}}, \quad (1.42)$$

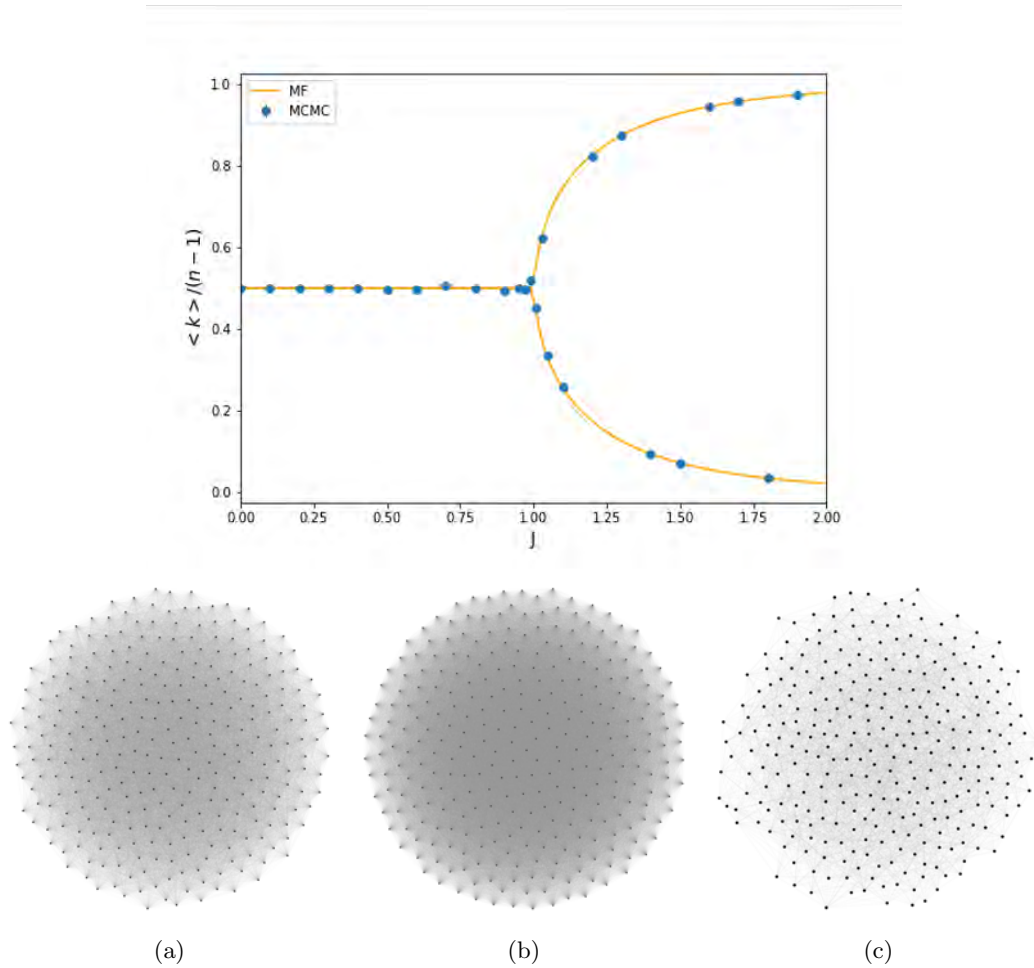


Figure 1.1: Connectance plot as a function of J with $\alpha = -J$. MF denotes mean-field solution and MCMC denotes the result of my simulation. The bifurcation indicate a phase transition at $J = 1$. The graphs (a), (b), and (c) in the second row are quantitative visualizations of two-star graphs at $J = 0.5$, $J = 1.5$ with high mean degree initial state, and $J = 1.5$ with low mean degree initial state. The symmetry around connectance = 0.5 is due to the edge-hole symmetry in the two-star Hamiltonian. In the graph visualizations, edges overlap with each other creating a visual opacity in the graph; the level of opacity can be roughly thought of as a density of edges of the corresponding vertex.

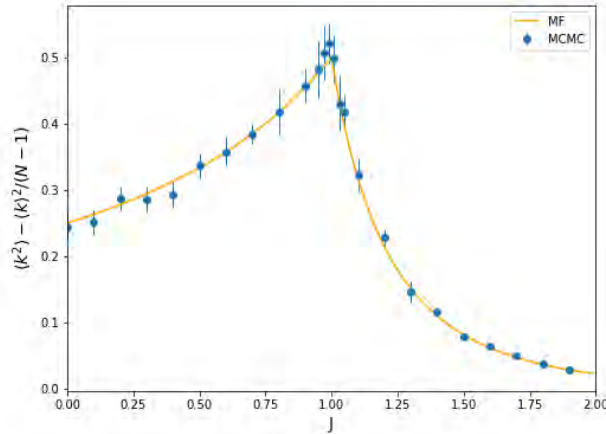


Figure 1.2: Variance of node degree as a function of J along the line $\alpha = -J$ at $N = 300$. The sharp peak at $J = 1$ indicates a phase transition. We observed that the fluctuations (reflected in the error bar) is larger in the region $J < 1$ than when $J > 1$.

where

$$g(k) \equiv \frac{(c \langle k \rangle)^{k/2} e^{-((k)+c)/2}}{k!}. \quad (1.43)$$

Here, c is an arbitrary constant in the order $\mathcal{O}(N^0)$. This solution also agree with the Erdős–Rényi graph for $\beta = 0$ [1].

Again, I numerically verified Annibale’s results [1]. In the sparse regime, we need to scale $\alpha \rightarrow \hat{\alpha} - 1/2 \log(N)$ to keep the connectivity in the order of $\mathcal{O}(N^0)$. I simulated the sparse regime for $\alpha = -0.5 - \frac{1}{2} \log N$ and $\alpha = 4 - \frac{1}{2} \log N$ with $N = 300$ and $c = 1$ in order to reproduce the result in Annibale’s work [1]. The outcome is as expected, the simulation agrees with (1.41) and (1.42), and the mean-field solution becomes inaccurate. Log connectivity ($\log \langle k \rangle / \log N$), log star-density ($\log (\langle k^2 \rangle - \langle k \rangle) / \log N$), and typical graph for both value of α is shown in Fig.1.3. We also present a two-star graph in Fig.1.4 whose mean degree is 6 that we would use as a baseline to our model later.

For low connectivity graph ($\alpha = -\frac{1}{2} - \log N$), the log connectivity and log star-density is negative meaning that there’s many nodes with no connection at all. The opposite is true in high-connectivity graph: both values are positive and the graph has much more connection overall.

I only show the result of $\beta < 0$ since we already explore $\beta > 0$ in the dense regime and large β would result in a complete graph.

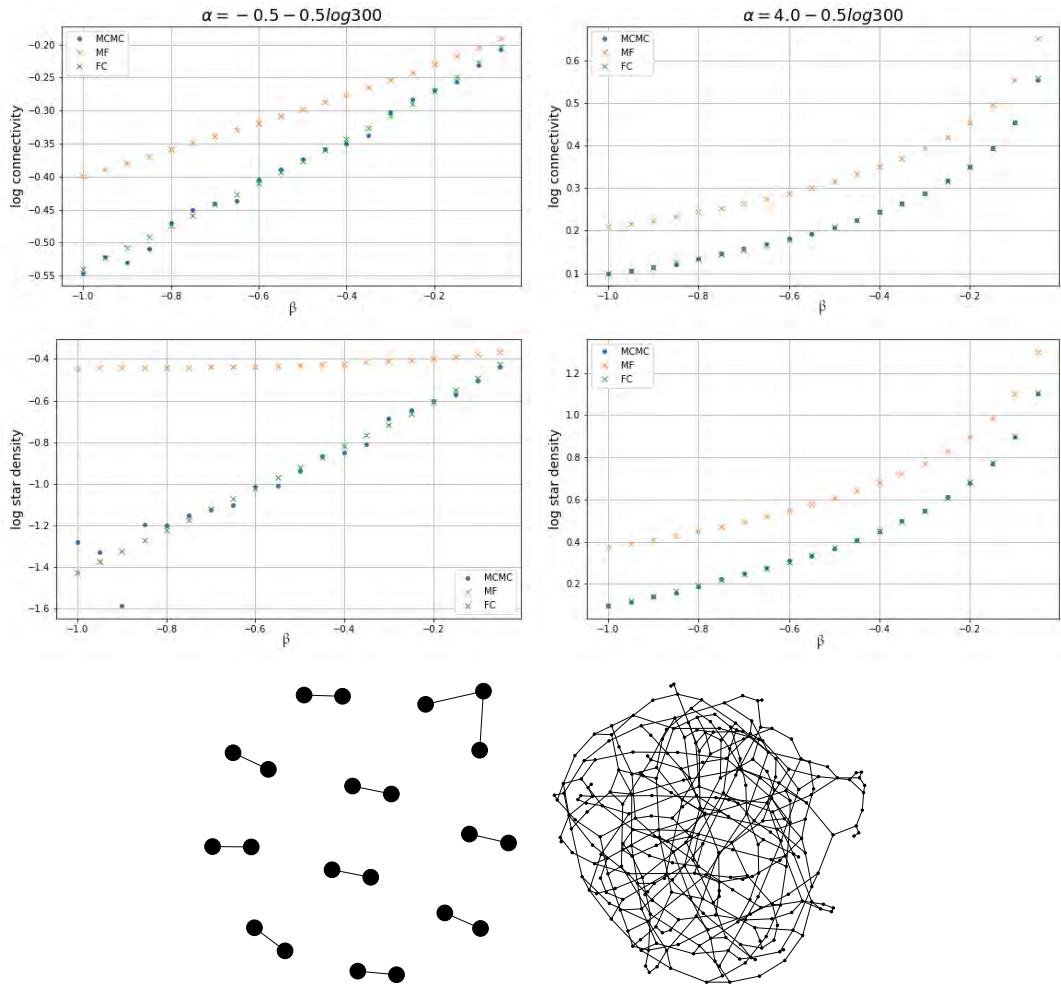


Figure 1.3: Plot of log connectivity ($\log \langle k \rangle / \log N$) and log star-density ($\log (\langle k^2 \rangle - \langle k \rangle) / \log N$) as a function of β for $\alpha = -0.5$, and $4 - \frac{1}{2} \log N$ with $N = 300$. MCMC denotes the result from the simulation, MF denotes mean-field solution, and FC denotes finite connectivity solution from Annibale's work (1.41), (1.42). On the last row, we have a visualization of a graph at $\alpha = -0.5 - \frac{1}{2} \log 300$, $\alpha = 4 - \frac{1}{2} \log 300$ to illustrate typical configurations in this sparse regime. Nodes with degree 0 are not shown in the visualization.

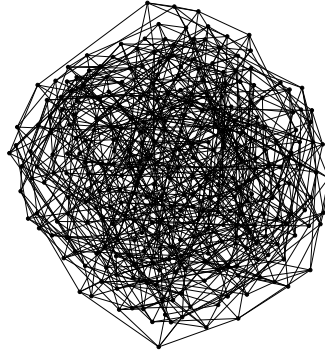


Figure 1.4: A two-star graph at $\alpha = -2.852$, $\beta = -10$, and $N = 300$. This graph with mean degree 6 and degree variance exactly 0 will be used as a baseline for non-geometric graph to be discussed further in Chapter 3.

1.4.3 Graph Collapse Problem (Degeneracy Problem)

Apart from our simulation results along the line $\alpha = -J$, almost all graphs in the two-star model are either very sparse or very dense. This problem is known as *degeneracy problem* in the context of ERGM where it prevent sociologists from finding a unique set of parameters to describe a real-world network [9]. It also discourages reasonable graphs of finite mean degree since the degree in the sparse region scale like $\mathcal{O}(N^{-1})$. This is not unique to the two-star model. In fact, the degeneracy problem is common to the family of ERGM. Introducing naive geometric graph motifs can easily cause the model to collapse into either of these two phases. This is shown in the Strauss' model of a clustered network where an introduction of a triangle term cause the graph to collapse into a sparse phase and a dense phase, where the triangles cluster together instead of distributing evenly over the graph [7]. This degeneracy problem is the main obstacle for traditional ERGMs to model reasonable social networks or to give rise to emergent discrete geometric primitives, which is the focus of this thesis.

Chapter 2

Geometric Graph Models

Our goal is to generate a graph ensemble that has a finite degree with small degree variations, also with a large number of geometric primitives, such as triangles and squares. To this end, we start with the two-star model with a large negative two-star parameter β , since the number of two-star relates closely with degree variance, and modify it by adding a geometric motif term to encourage such geometric structure. We can then control the mean degree by adjusting a degree parameter α . However, it is evident that we cannot naively put whatever geometric graph motif terms we want in the Hamiltonian since they can easily lead to the graph collapse problem. In this chapter, we shall describe our solution to the graph collapse problem and its results.

2.1 The Modified Square Model

Inspired by the *hard-core* constraints (used in [3]), which constrains two squares to only share an edge at most, we came up with our modification of the two-star model where we add the following term to the Hamiltonian:

$$\sigma \sum_{i>j} \delta(q_{ij}, 2), \quad (2.1)$$

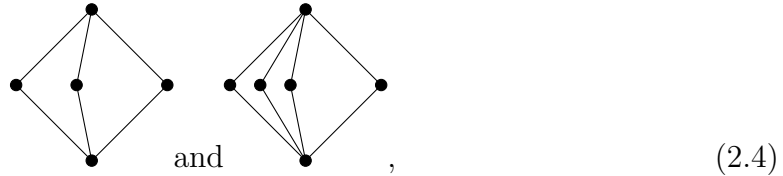
where δ is the Kronecker symbol, σ controls the strength of this term of this term, and q_{ij} is the square of the adjacency matrix

$$q_{ij} = \sum_k A_{ik} A_{kj}. \quad (2.2)$$

(2.1) term encourages subgraphs that have exactly 2 path of length 2, visualized as



which can be thought of as a squares. Additionally, (2.1) discourages subgraphs that have any other number of paths of length 2 such as



so that the squares do not concentrate on the same set of nodes, thus preventing the graph collapse problem. Our Hamiltonian is now in the form

$$H_{\square} = \underbrace{\left[-\alpha \sum_i k_i - \beta \sum_i k_i^2 \right]}_{\text{two-star Hamiltonian}} - \sigma \sum_{i>j} \delta(q_{ij}, 2) \quad (2.5)$$

We expect to see a large number of squares from this model in the sparse regime where the degree is N -independent. These geometric structures are expected to form a 3-dimensional square lattice up to defects when the mean degree is around 6.

2.2 The Modified Triangle Model

We can apply the same idea as the modified square model to the modified triangle model by introducing the term

$$\sigma \sum_{i>j} \delta(A_{ij}q_{ij}, 2). \quad (2.6)$$

Here, we element-wise multiply q_{ij} by A_{ij} so that it encourage the following structure



which can be thought of as 2 triangles glued together, instead of a single square. Likewise, this term prevents clustering of triangles seen in the Struass' model and should resolve the graph collapse problem by discouraging the structure like



The Hamiltonian of the modified triangle model is then given as

$$H_{\Delta} = \underbrace{\left[-\alpha \sum_i k_i - \beta \sum_i k_i^2 \right]}_{\text{two-star Hamiltonian}} - \sigma \sum_{i>j} \delta(A_{ij}q_{ij}, 2). \quad (2.9)$$

Similar to the modified square model, we expect to see a large number of triangles and partial hexagonal lattice when the mean degree is close to 6.

Chapter 3

Numerical Results

We simulate the modified triangle and the modified square model, which exhibit 6-regular graphs (the degree is approximately 6 with small degree variance). This choice of mean degree is motivated by the degree of nodes in triangular and simple cubic lattice. The results from our simulations suggest that both models that we proposed has achieved our goals; they form a large number of triangles and squares as well as other geometric structures, as we will see shortly. On top of that, they do not suffer from the graph collapse problem that plagues the standard ERGM family.

3.1 The Modified Triangle Model

To quantify the notion of “large” number of geometric primitives, we introduce the triangle fraction η_{Δ} defined as the number of triangle n_{Δ} , which is given by $Tr(\mathbf{A}^3)$ [11], divided by the maximum number of triangles of a d -regular graph with N nodes, given by $Nd(d-1)/6$. For a 6-regular graph, this quantity is equal to $5N$ so we can write

$$\eta_{\Delta} = \frac{n_{\Delta}}{5N}. \tag{3.1}$$

We also count the number of hexagons n_{hex} , which is the emergent structure we expect to see at mean degree approximately 6. This motif is defined as subgraphs in the form



with the condition that the central node strictly has degree 6 and the peripheral nodes do not have any other connections to each other. This means that the central node cannot connect to any other node outside the hexagon while the peripheral nodes

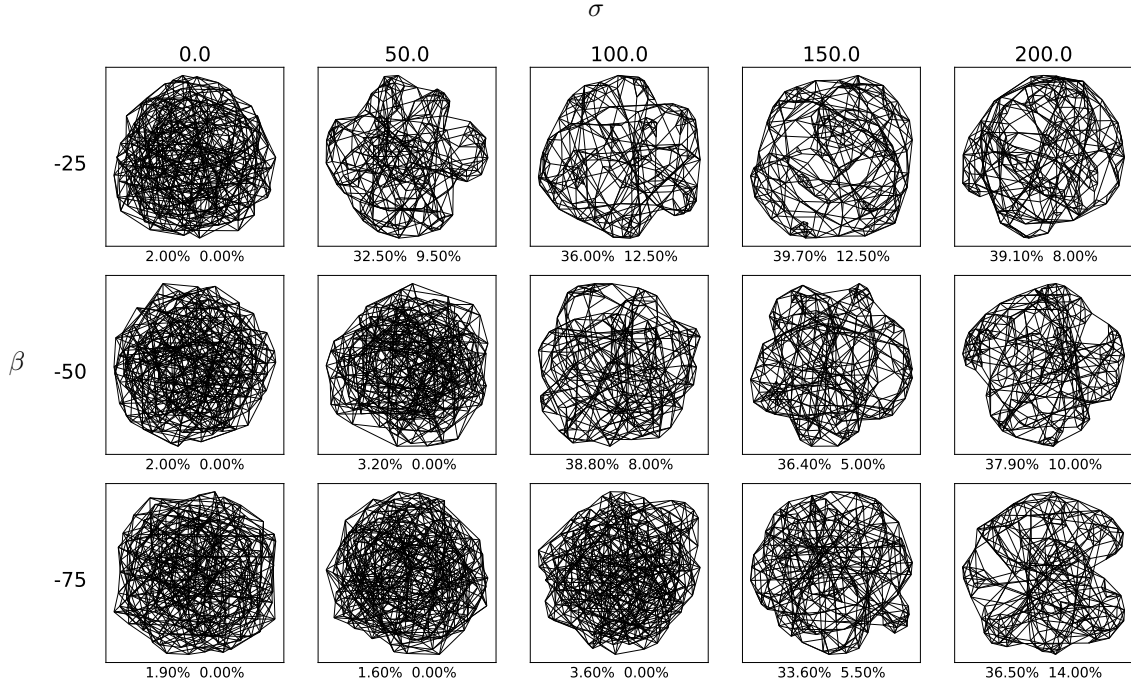


Figure 3.1: Typical graph configurations sampled from the modified triangle model plotted over the parameter space β and σ . α is tuned so that the mean degree is close to 6 (the range of mean degree in this plot is ± 0.1). The two percentages under each graph are the percentage of triangles and hexagons respectively.

can. We introduce the hexagon fraction

$$\eta_{hex} = \frac{n_{hex}}{N} \quad (3.3)$$

where we choose to scale it with the maximum number of hexagons which is the total number of nodes N since every node can be a center of a hexagon.

We begin our examination of the modified triangle model by making what we call a *scan plot* where we simulate several graphs of a small size to roughly explore the parameter space. Such plot is shown in Fig.3.1. We observe that there are 2 apparently distinct regimes of the typical graph configurations. When σ is not sufficiently large, the graph crumples up into a ball like what we observe in Fig.1.4. The two-star graph shown in Fig.1.4 has the percentage of triangles and hexagons comparable to that of the graphs below the diagonal of Fig.3.1. This suggests that the two-star term dominates when σ is small. When σ is sufficiently large, above the diagonal, the graph is more transparent and crystalline. We call this regime a *geometric graph phase*. The graphs above the diagonal also have a large percentage of triangles and a non-negligible percentage of hexagons. Note that these percentages are huge compared to sparse Erdos-Renyi graph, in which the percentage of triangles

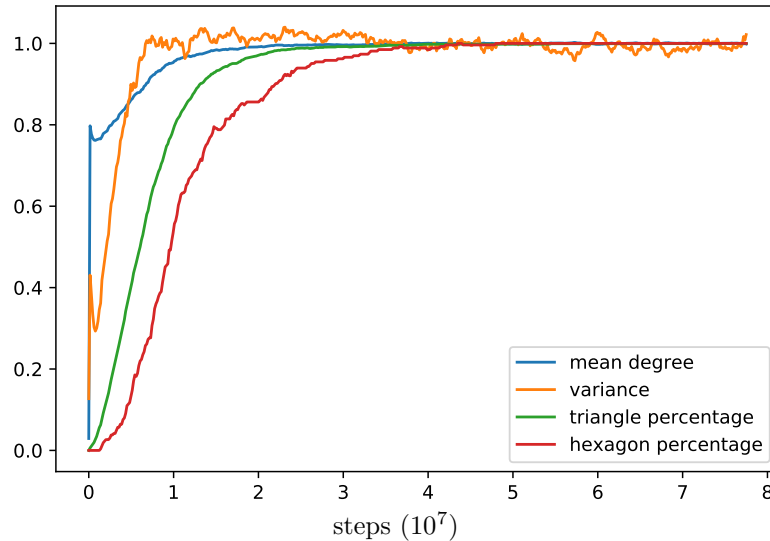


Figure 3.2: Convergence of the mean degree, degree variance, and the percentage of triangles, and of hexagons from the modified triangle model at $\alpha = 177$, $\beta = -20$, and $\sigma = 100$. Each quantity is re-scaled by its equilibrium value so that they are all normalized to 1. The horizontal axis is the Monte Carlo step count in units of 10 million steps.

goes to zero in the limit of large N , let alone the percentage of hexagons. Though we deliberately plot very small graphs, this picture is qualitatively correct up to the biggest graph we were able to simulate ($N = 5000$).

The next point we would like to address is the convergence of our simulation. According to Fig.3.2, the mean degree $\sum_i k_i/N$ is the first to converge followed by the mean degree variance $\sum_i k_i^2/N - (\sum_i k_i/N)^2$, the percentage of triangles, and the percentage of hexagons, respectively. After several simulations at different graph sizes, we estimate that the steps of Monte Carlo needed for the convergence of hexagons grow roughly like $\mathcal{O}(N^{2.8})$.

The most notable features of our model is its percentage of geometric primitives. Fig.3.3 shows that all ranges of density of the initial state sampled from an Erdos-Renyi graph consistently give large percentage of triangles and hexagons, although they end up in quite different states. This behavior that different initial states evolve into different final states and gets stuck there is known as *jamming*, which is also observed in other more conventional disordered systems, such as in glassy materials where the crystallization crucially depends on the initial configurations. Jamming can occur when the system is trapped in a spurious meta-stable state in a high-dimensional rugged energy landscape where it needs large energy fluctuations to get out. It is not surprising that the modified triangle model displays jamming behavior since it is difficult to obtain a perfect crystalline order in the final state when small

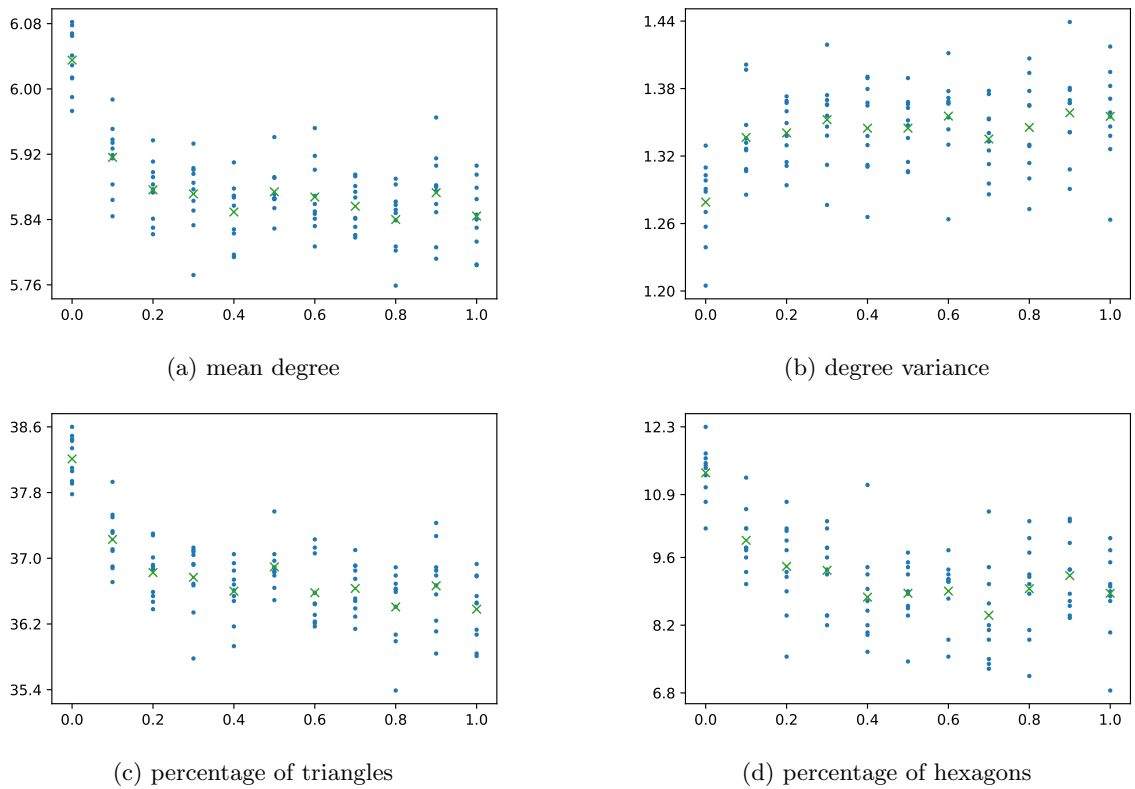


Figure 3.3: The mean degree, degree variance, and the percentage of triangles and of hexagons of the modified triangle model from Monte Carlo simulation at $N = 2000$, $\alpha = 177$, $\beta = -20$, $\sigma = 100$. The horizontal axes is the density of Erdos-Renyi graph initial state. The green crosses are mean values for each initial condition averaged over 10 Monte Carlo runs.

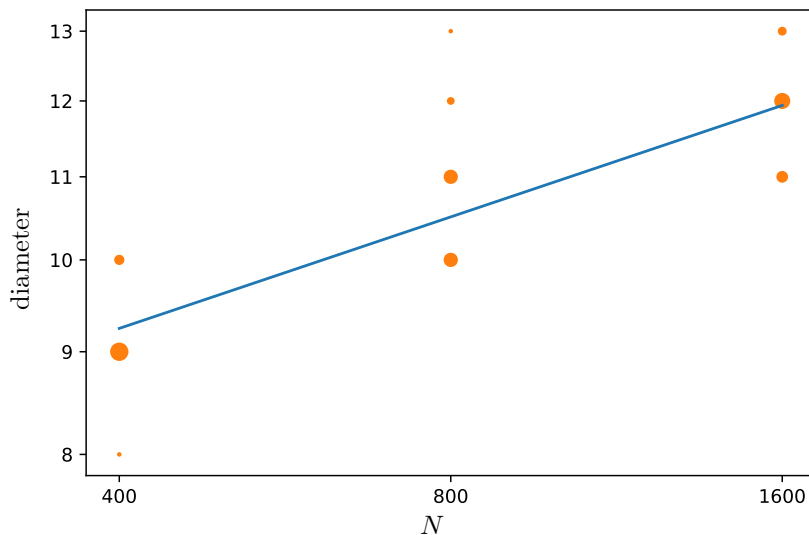


Figure 3.4: Diameter values of graphs in the modified triangle model plotted as a function of graph size N . At each size, 50 samples are simulated. The size orange dots represents the frequencies of data at each value of the diameter. The blue line is the best fit line, which gives the scaling law of the diameter $N^{0.1845 \pm 0.0083}$. This implies that the structures of graphs from the modified triangle model are more similar to lattice than Erdős–Rényi graph.

nucleations emerge simultaneously in various components of the graph. Indeed, once a large number of triangles, favored by the Hamiltonian, has formed in our graph, it is difficult to modify this configuration without paying a high energetic cost to “unjam” the configuration. Nevertheless, all of the final graphs, regardless of their initial states, exhibit significantly larger numbers of geometric primitives than any other ERGMs without hard constraints and we consider this as our success!

We also fit diameter values of several graph sizes to get a scaling law for the diameter. It is known that the diameter of sparse Erdos-Renyi graph scales as the logarithm of its size [6]. However, the best fit line of the diameter of the modified triangle model in Fig.3.4 suggests that it grows as $N^{0.1845 \pm 0.0083}$. This result suggests that our graph is more similar to a regular lattice or a simplicial complex, where the diameter grows as the power of N , than to a traditional random graph.

Last but not least, we present the largest graph we are able to simulate within our computational capability with $N = 5000$ at $\alpha = 177$, $\beta = -20$, and $\sigma = 10$ to give a visual impression in Fig.3.5.

3.2 The Modified Square Model

Similar to the modified triangle model, we keep track of the number of the squares n_{\square} , which is given by $(Tr(\mathbf{A}^4) + Tr(\mathbf{A}^2) - 2 \sum_i k_i^2)/8$ [11], in the form of square

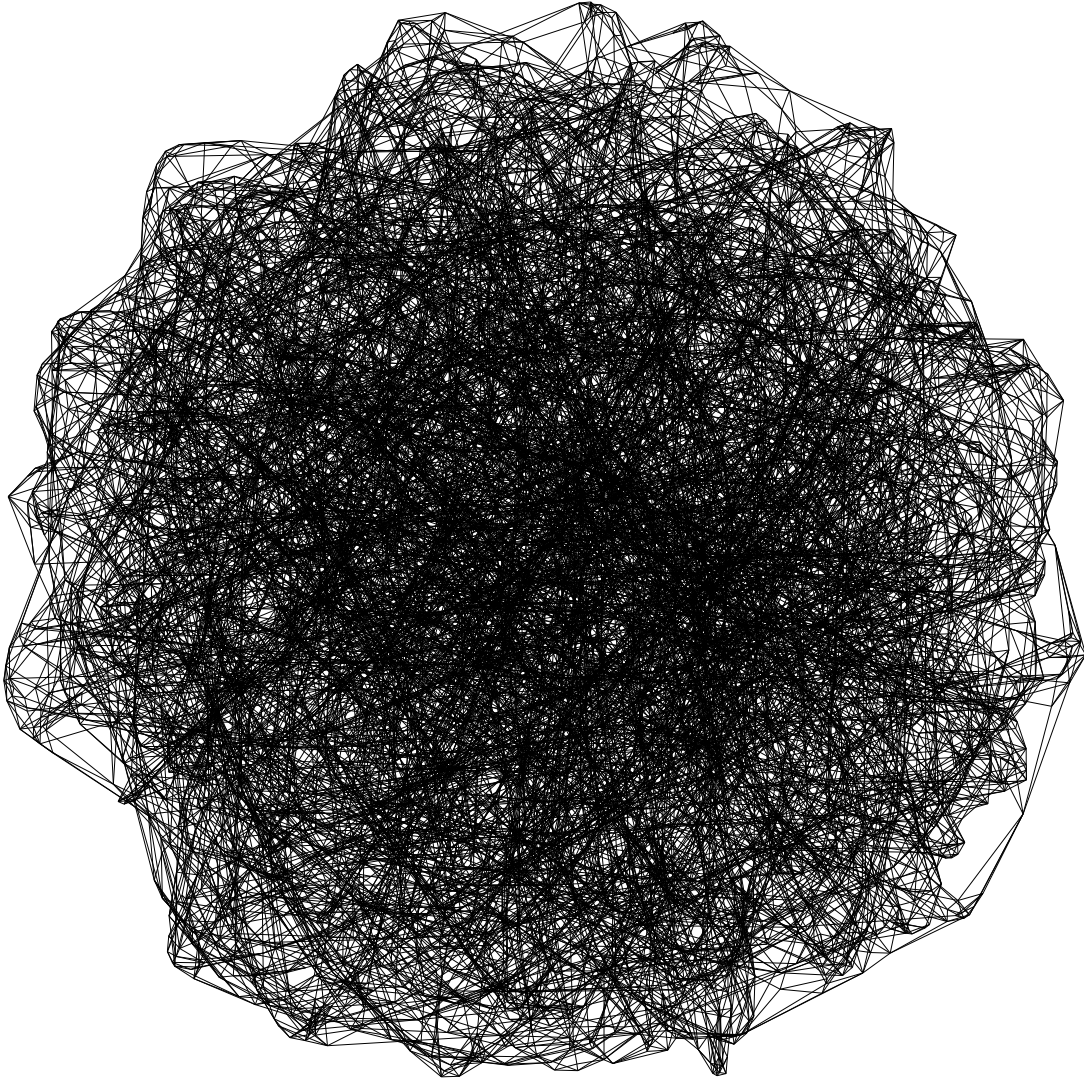


Figure 3.5: A very large modified triangle graph obtained from the simulation at $N = 5000, \alpha = 177, \beta = -20, \sigma = 100$.

fraction η_{\square} defined as

$$\eta_{\square} = \frac{n_{\square}}{15N}, \quad (3.4)$$

where the denominator is now the maximum number of squares in a 6-regular graph, given by $Nd(d-1)(d-2)/8$ which is $15N$. We also count the number of cubes n_{cube} , which is strictly defined as a subgraph of the form

$$, \quad (3.5)$$

as a counterpart of the number of hexagons in the modified triangle model. This cube subgraph has no other connection between each of its member but can connect

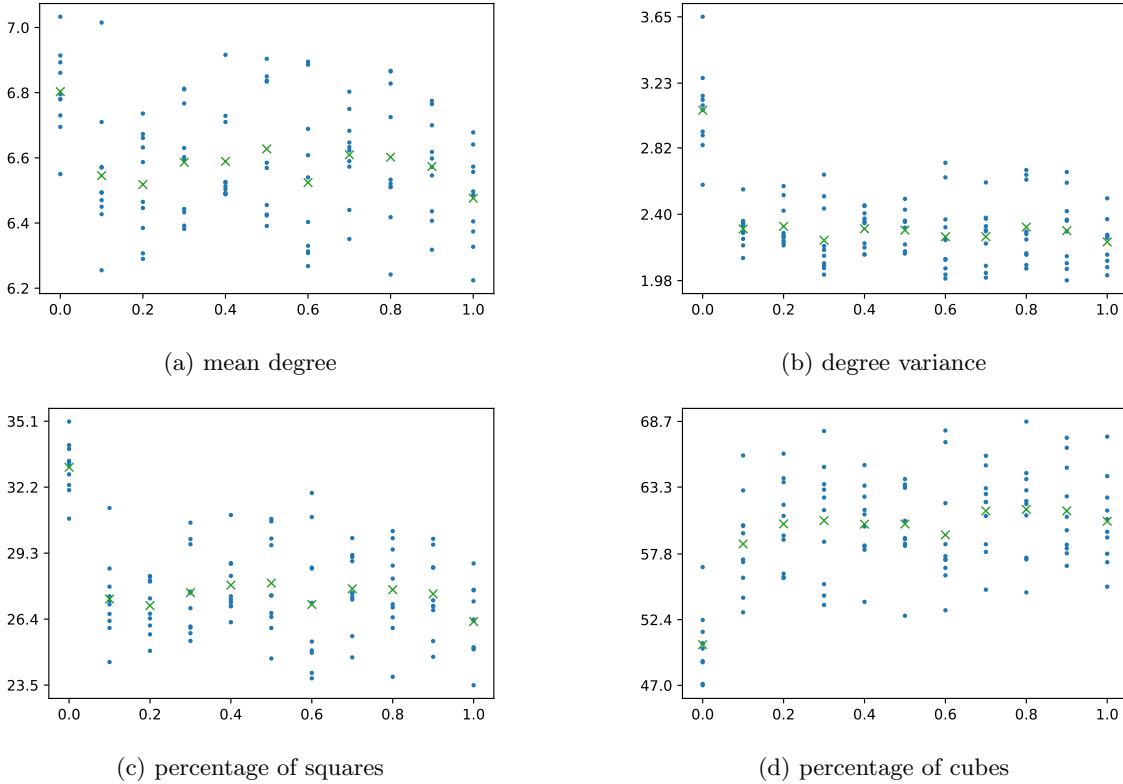


Figure 3.6: The mean degree, degree variance, and the percentage of squares and of cubes of the modified square model from Monte Carlo simulation at $N = 2000$, $\alpha = 80$, $\beta = -20$, $\sigma = 100$. The horizontal axes is the density of Erdos-Renyi graph initial state. The green crosses are mean values for each initial condition averaged over 10 Monte Carlo runs.

to any other node in the graph. The cube fraction is defined like its modified triangle model counterpart

$$\eta_{cube} = \frac{n_{cube}}{N}. \quad (3.6)$$

Overall, the behavior of the modified square model is very similar to the modified triangle model. When σ is sufficiently large, it gives large a percentage of squares and cubes. It also displays a similar jamming behavior as the modified triangle model although the variation of the jammed final state is larger than that of the modified triangle model. We summarize this in Fig.3.6 which is to be compared and contrasted with Fig.3.3.

Chapter 4

Conclusion

In this thesis, we begin by reviewing the statistical mechanics framework of Exponential Random Graph Model (ERGM), a family of random graph models in which graph ensembles are prescribed by the Hamiltonian in the Boltzmann distribution. We also review the analytic solutions of the two-star model, the simplest model in ERGM family that displays phase transitions. The analytic treatments of the two-star model, studied extensively in [1, 8], have been verified by our Monte Carlo simulations.

The second part of this thesis consists of our efforts to design a random graph with a significant number of geometric primitives without suffering from a graph collapse phenomenon. We come up with a modified triangle and a modified square model, where we introduce a new term (A.14) and (A.13) to the Hamiltonian of the two-star model. In the parameter regime where these new terms are large, the graph is populated by significant fractions of geometric primitives including triangle, hexagons, squares, and cubes. Since we do not introduce a cube or a hexagon-counting term to the Hamiltonian, these geometric features of our models are considered to be emergent property. Comparing to the original Erdos-Renyi graph where these motifs are expected to vanish in the limit of large N , we consider our model as a success. Other notable features of our models include their jamming behaviors, in which the equilibrium of both modified triangle and square models displays an initial state dependency. In addition, they also show non-geometric to geometric graph phase transition shown in Fig. 3.1. These topics require intensive analytical treatments and are out of the scope of this thesis. To this end, we see our work as a first step to investigate emergent geometry from random graph models.

Appendix A

Numerical Methods

One of the common methods to simulate a graph ensemble is to use a Metropolis-Hastings algorithm, a class of Markov Chain Monte Carlo (MCMC) algorithm. In short, a graph from any chosen initial state will be updated stochastically via a Markov process and is guaranteed to reach the desired equilibrium ensemble by detail balance and ergodicity condition. This section aims to review the Metropolis Monte Carlo algorithm based on the book by Newman and Barkema [5] and shows how one can optimize this algorithm.

A.1 Markov Chain Monte Carlo

In this section, we review background concepts and Metropolis Monte Carlo algorithm. Monte Carlo algorithm is known as a brute force solution to hard numerical problems. It uses a random number generator to mimic probabilistic properties of physics systems. With a proper setup, it can be a powerful tool for various problems.

The system of our interest is the Boltzmann distributed graph ensemble. The obvious problem we face is that this distribution is dominated by small numbers of states around the minimum(s) of the Hamiltonian. To sample from this distribution, we use Metropolis algorithm. This algorithm makes use of Markov Chain with some constraints to create auxiliary dynamics that ends up with Boltzmann distributed states in the equilibrium. In this section, we shall discuss each element of this algorithm.

A.1.1 Markov Chain

A Markov Process is defined as a process to randomly generate a new state ν from a current state μ with the time independent probability that only depends on μ and ν called *transition probability* $P(\mu \rightarrow \nu)$. Clearly, transition probability has to satisfy

the condition

$$\sum_{\nu} P(\mu \rightarrow \nu) = 1 \quad (\text{A.1})$$

so that the overall out-going probability is normalized.

Markov chain is a chain of Markov processes. We can choose a specific Markov chain that eventually gives a successive chain of Boltzmann distributed states. The constraints we shall impose on the Markov chain is called *detailed balance* and *ergodicity*.

A.1.2 Detailed Balance

This condition ensures that we'll reach the desire distribution when Markov Chain reach its equilibrium. At the equilibrium, the rate that the system evolves to μ and from μ must be equal so one can write

$$\sum_{\nu} p_{\mu} P(\mu \rightarrow \nu) = \sum_{\nu} p_{\nu} P(\nu \rightarrow \mu) \quad (\text{A.2})$$

or simply

$$p_{\mu} = \sum_{\nu} p_{\nu} P(\nu \rightarrow \mu). \quad (\text{A.3})$$

In a naive sense, at the equilibrium, the probability of every state should stay the same, or in other words, be in the steady state. There is, however, a more general kind of equilibrium known as *dynamic equilibrium* where the states evolve in a circular manner called *limit cycle*. The limit cycle is not desirable since it shifts the probability distribution of our system away from what we want. One way to solve this problem is to introduce a *detailed balance* condition

$$p_{\mu} P(\mu \rightarrow \nu) = p_{\nu} P(\nu \rightarrow \mu). \quad (\text{A.4})$$

This condition states that the system should be arriving at the state μ as likely as departing from it. This condition is satisfied in case of simple equilibrium but in limit cycle, where states tend to evolve with the cycle, this condition is violated.

From (A.4), given we want our system to end up in the equilibrium Boltzmann distribution, we can write

$$\frac{P(\mu \rightarrow \nu)}{P(\nu \rightarrow \mu)} = \frac{p_{\nu}}{p_{\mu}} = e^{-(E_{\nu} - E_{\mu})}. \quad (\text{A.5})$$

A.1.3 Ergodicity

Ergodicity is the condition that requires all possible states to be reachable by the Markov process. This guarantees that every initial state will be able to reach the equilibrium and no possible states will miss out so that we end up with Boltzmann distribution in the equilibrium. Note that this does not mean that every state has to be able to reach every other one, only that they should be able to reach one by at least one path of non-zero overall probability. The proof of ergodicity of our algorithm is out of the scope of this thesis and thus will be left out.

A.1.4 Acceptance Ratios

In practice, it is not trivial to generate a Markov processes which has the transition probability $P(\mu \rightarrow \nu)$. We instead use two separate process $g(\mu \rightarrow \nu)$ and $A(\mu \rightarrow \nu)$.

$$P(\mu \rightarrow \nu) = g(\mu \rightarrow \nu)A(\mu \rightarrow \nu) \quad (\text{A.6})$$

Here, $g(\mu \rightarrow \nu)$ is used to generate a new state from the old one. $A(\mu \rightarrow \nu)$ is used to accept it.

A.1.5 The Metropolis Algorithm

From the previous section, we can see that the essence of Markov Chain Monte Carlo is to choose a proper generating probability $g(\mu \rightarrow \nu)$ and acceptance rate $A(\mu \rightarrow \nu)$. For the Metropolis algorithm, $g(\mu \rightarrow \nu)$ is the transition probability from μ to a slightly different state. In this case, this is the probability to flip a single element of the adjacency matrix which is fixed to one over the size of the adjacency matrix. With this, (A.5) turns to

$$\frac{A(\mu \rightarrow \nu)}{A(\nu \rightarrow \mu)} = e^{-(E_\nu - E_\mu)}. \quad (\text{A.7})$$

All we have to do now is to choose an appropriate acceptance rate. It is wise to choose $A(\mu \rightarrow \nu)$ as high as possible and the other one according to the constraints so the system gets to explore more states. We also observe that if the state ν has lower energy than μ , the exponent of (A.5) is positive so the system tends to evolve toward lower energy. Thus, one way to get the most efficiency out of the Markov Chain Monte Carlo, the Metropolis algorithm suggests to choose

$$A(\mu \rightarrow \nu) = \begin{cases} e^{-(E_\nu - E_\mu)} & \text{if } E_\nu - E_\mu > 0 \\ 1 & \text{otherwise.} \end{cases} \quad (\text{A.8})$$

The non-zero probability to flip to a higher-energy state also allows our algorithm to escape local minima in the Hamiltonian.

A.2 Optimization of The Modified Model

In essence, the algorithm of the modified triangle or the square model and the two-star model are the same. However, we need to optimize our code since now our Hamiltonian has more time complexity; directly computing the number of two-stars has the complexity of that of matrix multiplication but finding the number of 2 length-2-paths involve several matrix multiplications and searching. The goal of this section is to derive the formula relating the change of the Hamiltonian at each step to the change of the adjacency matrix. After that, we shall get to the results.

At each step, the ij element of the adjacency matrix is flipped from 0 to 1 or 1 to 0. This change of the element ij can be written as

$$\Delta_{ij}c_{mn} = \delta_{im}\delta_{jn} + \delta_{in}\delta_{jm}(1 - 2c_{ij}). \quad (\text{A.9})$$

Recall that degree is defined as $k_m = \sum_n c_{mn}$ so the change of the degree due to ij -flip is

$$\Delta_{ij}k_m = (\delta_{im} + \delta_{jm})(1 - 2c_{ij}). \quad (\text{A.10})$$

Substituting this into the Hamiltonian of the two-star model (1.9) gives

$$\Delta_{ij} = -2(1 - 2c_{ij})[\alpha + \beta(k_i + k_j)] + 2\beta. \quad (\text{A.11})$$

For the modified square model, the change of q is given as

$$\Delta_{ij}q_{mn} = (\delta_{im}c_{jn} + \delta_{jm}c_{in} + \delta_{jn}c_{im} + \delta_{in}c_{jm})(1 - 2c_{ij}) + (\delta_{im}\delta_{in} + \delta_{jm}\delta_{jn}). \quad (\text{A.12})$$

the modified triangle model has a more complex formulae

$$\Delta_{ij}(c_{mn}q_{mn}) = q_{mn}(\Delta_{ij}c_{mn}) + c_{mn}(\Delta_{ij}q_{mn}) + (\Delta_{ij}c_{mn})(\Delta_{ij}q_{mn}). \quad (\text{A.13})$$

However, $\Delta_{ij}c_{mn}$ is only non-zero if $m = i$ and $n = j$ or $m = j$ and $n = i$, in these cases, $\Delta_{ij}q_{mn}$ is zero so the last term vanishes. The term $c_{mn}(\delta_{im}\delta_{in} + \delta_{jm}\delta_{jn})$ from the second term also vanishes since the diagonal elements of our adjacency matrix is zero. Thus (A.13) can be rewritten as

$$\Delta_{ij}(c_{mn}q_{mn}) = q_{mn}(\delta_{im}\delta_{jn} + \delta_{in}\delta_{jn})(1 - 2c_{ij}) + c_{mn}(\delta_{im}c_{jn} + \delta_{jn}c_{in} + \delta_{jn}c_{im} + \delta_{in}c_{jm})(1 - 2c_{ij}). \quad (\text{A.14})$$

Note that both (A.13) and (A.14) only make changes to rows and columns i and j so that we don't have to multiply entire matrices every step. Thus, the time complexity of our algorithm is reduced to only $\mathcal{O}(N)$.

Appendix B

Code

In this appendix, we describe our program used to sample our exponential random graph ensemble. The program is written in Python with package Numpy, Matplotlib, and Scipy.

B.1 The Erdős–Rényi Graph

At its essence, the Erdős–Rényi graph is a graph with fixed amount of nodes randomly connected by a fixed amount of edges. This means we can create an adjacency matrix of Erdős–Rényi graph by assigning 1 to each component of a zero matrix with a probability equal to *graph density*, defined to be the ratio of the total number of edges to the total number of nodes. Then, we can make it undirected by mirroring the components either upper or below the diagonal. The code for generating an Erdős–Rényi graph is the following:

```
1 def ini_ER (N,p):
2     state = np.triu((np.random.rand(N,N)<p).astype('float64'))
3     np.fill_diagonal(state,0)
4     return state+np.transpose(state)
```

B.2 The Two-star Model

Unlike the Erdős–Rényi graph, the distribution of the two-star ensemble has to be simulated by MCMC algorithm. I divided my program into 3 parts: initialization, Markov process, and outputs. The last one depends on styles and individual preferences so I will not get into the specific details. This section will focus on the first and the second part of my program. The full code is shown at the end of this section.

B.2.1 Initialization

When working with MCMC algorithm, it is wise to use the initial state similar to the expected results, if possible, to avoid falling into spurious local minima. In the two-star program, we've experimented with various initial states. An empty graph can be easily generated using `zeros` function from Numpy:

```
1 np.zeros((N,N))
```

We can also create a graph with fixed degree K using the following code:

```
1 def ini_fixed(N,K):
2     state = np.zeros((N,N), dtype = 'int')
3     set_ij = [[j,i] for i in range(1,N) for j in range(i)]
4     for ij in random.sample(set_ij,k = K):
5         state[ij[0],ij[1]] = 1
6
7     for i in range(N):
8         for j in np.random.choice(N,size=K,replace=False):
9             state[j,i] = 1
10    for i in range (N):
11        for j in range(N):
12            if (i==j):
13                state[j,i] = 0
14            elif (i>j):
15                state[i,j] = state[j,i]
16    return state
```

Finally, the following code is used to create a random graph with half the edge possible:

```
1 def ini_random(N):
2     state = np.random.randint(2,size = (N,N))
3
4     for i in range (N):
5         for j in range(N):
6             if (i==j):
7                 state[j,i] = 0
8             elif (i>j):
9                 state[i,j] = state[j,i]
10
11    return state
```

B.2.2 Markov Process

The function associated to MCMC is called `flip`. This function updates the adjacency matrix with the desired move, calculates the energy difference, and accepts the move according to the metropolis algorithm described in appendix A. The moves implemented includes, normal flip, hinge flip, edge swap, big flip, and column flip. Hinge flip move preserves mean degree and edge swap preserves both mean degree and the degree of affected nodes. Big flip and column flip updates a large portion of the

adjacency matrix and a column of the matrix respectively; they are useful strategies to escape spurious local minima, if used correctly.

B.2.3 Full Code

```

1 def cal_energy (state,alpha,beta):
2     k_0 = np.sum(state,axis=0)
3     k2_0 = np.power(k_0,2)
4     return -beta*np.sum(k2_0) -alpha*np.sum(k_0)
5
6 def normal_flip(N,state,alpha,beta):
7     E_0 = cal_energy(state,alpha,beta)
8
9     i = np.random.randint(1,N)
10    j = np.random.randint(0,i)
11
12    state[j,i] = 1-state[j,i]
13    state[i,j] = state[j,i]
14
15    E_n = cal_energy(state,alpha,beta)
16
17    if not(np.random.random() < min(1,np.exp(E_0-E_n))):
18        state[j,i] = 1-state[j,i]
19        state[i,j] = state[j,i]
20        return E_0
21
22    return E_n
23
24 def column_flip(N,state,alpha,beta):
25    E_0 = cal_energy(state,alpha,beta)
26
27    i = np.random.randint(0,N)
28
29    state[:,i] = 1-state[:,i]
30    state[i,:] = 1-state[i,:]
31
32    E_n = cal_energy(state,alpha,beta)
33
34    if not(np.random.random() < min(1,np.exp(E_0-E_n))):
35        state[:,i] = 1-state[:,i]
36        state[i,:] = 1-state[i,:]
37        # print('not column flipped')
38        # print(E_0,E_n)
39        return E_0
40
41    return E_n
42
43 def big_flip(N,n,state,alpha,beta):
44    E_0 = cal_energy(state,alpha,beta)
45    state0 = state.copy()
46
47    set_ij = np.sort(np.random.randint(0,N,size = (n,2)))
48

```

```

49     for thing in set_ij:
50         i = thing[1]
51         j = thing[0]
52         if i!=j:
53             state[j,i] = 1-state[j,i]
54             state[i,j] = state[j,i]
55
56     E_n = cal_energy(state,alpha,beta)
57
58     if not(np.random.random() < min(1,np.exp(E_0-E_n))):
59         state = state0.copy()
60         return E_0
61
62     return E_n
63
64 def edge_swap(N,state,alpha,beta):
65     E_0 = cal_energy(state,alpha,beta)
66
67     set_i = np.random.randint(1,high = N,size=2)
68     i1 = int(set_i[0])
69     j1 = np.random.randint(0,high = i1)
70     i2 = int(set_i[1])
71     j2 = np.random.randint(0,high = i2)
72
73     temp = state[j1,i1]
74     state[j1,i1] = state[j2,i2]
75     state[j2,i2] = temp
76     state[i1,j1] = state[j1,i1]
77     state[i2,j2] = state[j2,i2]
78
79     E_n = cal_energy(state,alpha,beta)
80
81     if not(np.random.random() < min(1,np.exp(E_0-E_n))):
82         temp = state[j1,i1]
83         state[j1,i1] = state[j2,i2]
84         state[j2,i2] = temp
85         state[i1,j1] = state[j1,i1]
86         state[i2,j2] = state[j2,i2]
87         return E_0
88     return E_n
89
90 def hinge_flip(N,state,alpha,beta):
91     E_0 = cal_energy(state,alpha,beta)
92
93     i1 = np.random.randint(1,high=N)
94     j1 = np.random.randint(0,high = i1)
95     j2 = np.random.randint(0,high = i1)
96
97     temp = state[j1,i1]
98     state[j1,i1] = state[j2,i1]
99     state[j2,i1] = temp
100    state[i1,j1] = state[j1,i1]
101    state[i1,j2] = state[j2,i1]
102

```

```

103     E_n = cal_energy(state, alpha, beta)
104
105     if not(np.random.random() < min(1, np.exp(E_0-E_n))):
106         temp = state[j1, i1]
107         state[j1, i1] = state[j2, i1]
108         state[j2, i1] = temp
109         state[i1, j1] = state[j1, i1]
110         state[i1, j2] = state[j2, i1]
111
112         return E_0
113     return E_n
114
115
116
117 def flip(frames, state, N, alpha, beta, plot_t = True): #state2 has higher Temp
118     ydata = []
119     int_frames = frames
120
121
122     while(frames > 0):
123         E = normal_flip(N, state, alpha, beta)
124         if frames % int(int_frames/200) == 0:
125             k_0 = np.sum(state, axis=0)
126             ydata.append(np.mean(k_0))
127             frames -= 1
128         if plot_t:
129             plt.plot(ydata)
130             plt.show()
131
132 def ini_random(N):
133     state = np.random.randint(2, size = (N, N))
134
135     for i in range(N):
136         for j in range(N):
137             if (i==j):
138                 state[j, i] = 0
139             elif (i>j):
140                 state[i, j] = state[j, i]
141
142     return state
143
144 def ini_fixed(N, K):
145     state = np.zeros((N, N), dtype = 'int')
146     set_ij = [[j, i] for i in range(1, N) for j in range(i)]
147     for ij in random.sample(set_ij, k = K):
148         state[ij[0], ij[1]] = 1
149
150     for i in range(N):
151         for j in np.random.choice(N, size=K, replace=False):
152             state[j, i] = 1
153     for i in range(N):
154         for j in range(N):
155             if (i==j):
156                 state[j, i] = 0

```

```

157         elif (i>j):
158             state[i,j] = state[j,i]
159     return state
160
161 def p_eq(p, alpha, beta, N):
162     return 0.5*(np.tanh(2*beta*p + alpha)+1)-p
163
164 def from_adj (N, alpha, beta):
165     return np.loadtxt('adj matrix\\'+str(N)+'alpha'+str(alpha)+' finite_twostar'+str(
166         beta)+'.adj', dtype = 'int')
167
168 def save_adj(N):
169     K = np.loadtxt('alpha_newton N=500.txt')
170     for chosen_i in (74,):
171         alpha = K[chosen_i,0]
172         beta = K[chosen_i,1]
173         mean_deg = K[chosen_i,2]
174
175
176         print('alpha : %.4f, beta : %.4f, <k> = %.4f, N = %d'%(alpha, beta, mean_deg, N)
177     )
178
179     state1 = ini_fixed(N, int(mean_deg*N/2))
180
181     k_0 = np.sum(state1, axis=0)
182     print(np.mean(np.power(k_0, 2)))
183
184     flip(1e8, state1, N, alpha, beta)
185
186     k_0 = np.sum(state1, axis=0)
187     print(np.mean(np.power(k_0, 2)))
188     plt.show()
189     np.savetxt('saved matrix\\'+str(N)+'alpha'+str(alpha)+' finite_twostar'+str(
190         beta)+'.adj', state1)
191
192 def main (N, steps):
193     beta = np.append(np.arange(0, 0.95, 0.1), np.append(np.arange(0.95, 1.05, 0.02), np.
194         arange(1.1, 2, 0.1)))
195     alpha = -beta
196
197     FileName = ' test non-sparse N= %d.txt'%(N,)
198
199     file = open(FileName, 'w')
200
201     xdata = []
202     ydata1 = []
203     ydata2 = []
204
205     print('N =', N)
206     for i in range(len(beta)):
207
208         datum1 = []
209         datum2 = []

```



```

207
208 # =====
209 #           # Initialize adj
210 # =====
211     # state1 = np.zeros((N,N))
212     # state1 = ini_random(N)
213     state1 = ini_fixed(N, 40)
214     # state1 = from_adj(beta[i])
215     # state1 = np.float32(np.delete(np.delete(np.loadtxt('saved matrix\\'+str(N)
+'alpha'+str(alpha[i])+'twostar'+str(beta[i])+'.csv', dtype = str, delimiter = ','),
,0,0),0,1))
216
217
218     flip(steps, state1, N, alpha[i], beta[i]/(N-1))
219     for itet in (2e2,)*100:
220         flip(itet, state1, N, alpha[i], beta[i]/(N-1), False)
221         deg = np.sum(state1, axis = 0)
222         datum1.append(np.mean(deg))
223         datum2.append(np.mean(np.power(deg, 2)) - np.power(np.mean(deg), 2))
224
225     data1 = np.mean(datum1)
226     data2 = np.mean(datum2)
227     error1 = np.std(datum1)
228     error2 = np.std(datum2)
229     print (beta[i], data1, data2)
230     xdata = np.append(xdata, beta[i])
231     ydata1 = np.append(ydata1, data1)
232     ydata2 = np.append(ydata2, data2)
233     file.write(str(beta[i])+'\t'+str(data1)+'\t'+str(error1)+'\t'+str(data2)+'\t'
+str(error2)+'\n')
234
235     file.close()
236
237 def plot_data():
238     A = np.loadtxt('non-sparse w er N= 300.txt')
239     beta = A[:,0]
240     beta_num = np.delete(np.arange(0,2,0.01),100)
241     alpha = -beta
242     alpha_num = -beta_num
243     ydata1 = A[:,1]
244     ydata2 = A[:,3]
245     ydatanumer = []
246     ydatanumer1 = []
247     ydatanumer2 = []
248     N = 300
249     y_error1 = A[:,2]
250     y_error2 = A[:,4]
251
252     fig = plt.figure(figsize = (8,6))
253     ax = plt.axes()
254
255     x = np.arange(0,2,0.01)
256     plt.plot(x, [p_eq(i, -1.5, 1.5, N) for i in x])
257     plt.show()

```

```

258
259     for i in range(len(beta_num)):
260         p = scpo.newton(p_eq,0.5,args=(alpha_num[i],beta_num[i],N))
261         ydatanumer1.append((N-1)*p+2*beta_num[i]*p*(1-p)*(1-2*p)/(1-4*beta_num[i]*p
*(1-p))/(1-2*beta_num[i]*p*(1-p)))
262         # p = scpo.newton(p_eq,0,args=(alpha_num[i],beta_num[i],N))
263         # ydatanumer2.append((N-1)*p+2*beta_num[i]*p*(1-p)*(1-2*p)/(1-4*beta_num[i]*p
*(1-p))/(1-2*beta_num[i]*p*(1-p)))
264         #ydatanumer.append(p*(1-p)/(1-2*beta_num[i]*p*(1-p)))
265
266
267     plt.xlim(0,2)
268     # ax.set_ylim(-0.7,1)
269     # plt.errorbar(beta,(ydata2/(N-1)),yerr = y_error2/(N-1),fmt = 'o',elinewidth =
1,label = 'MCMC')
270     # plt.errorbar(beta,(ydata1/(N-1)),yerr = y_error1/(N-1),fmt = 'o',elinewidth=1,
label = 'MCMC')
271     plt.plot(beta_num,np.array(ydatanumer1)/(N-1),'orange', label = 'MF')
272     # plt.plot(beta_num,np.array(ydatanumer2)/(N-1),'orange')
273
274     plt.xlabel('J',fontsize = 14)
275     plt.ylabel('$\langle k \rangle / (n-1)$',fontsize = 14)
276     plt.legend()
277     # plt.savefig("non_sparse_test")
278     plt.show()
279
280 main(50,20000)
281 # plot_data()
282 #save_adj(500)

```

B.3 The Modified Two-star Model

At its core, the two-star and our modified model are almost the same. However, for the modified model, we decide to reduce the code to only its working part since we have to send many copies of it to run on cluster computers. With the optimizations done in appendix A.2, we can directly calculate the variation of the energy of each update allowing us to get rid of the function `cal_energy` and tidy up the code even more. The following code is used to generated the modified triangle model:

```

1 def ini_ER (N,p):
2     state = np.triu((np.random.rand(N,N)<p).astype('float64'))
3     np.fill_diagonal(state,0)
4     return state+np.transpose(state)
5
6 def flip(step, state,state2,sstate2,N,alpha,beta,sigma,plot_t = True):
7     ydata = [[],[ ]]
8
9     snap=step//200
10    track = 100000
11

```

```

12     deg = np.copy(np.diagonal(state2))
13     sstate2new=np.copy(sstate2)
14     state2new = np.copy(state2)
15     while(step > 0):
16         i = np.random.randint(1,N)
17         j = np.random.randint(0,i)
18
19         # degnew = np.copy(deg)
20
21         varij=1-2*state[i,j]
22
23         #update sstate2
24         sstate2new[j,i] += varij*state2[j,i]
25         sstate2new[i,j] = sstate2new[j,i]
26         vij=state[:,i]*state[:,j]*varij
27         sstate2new[:,i] += vij
28         sstate2new[:,j] += vij
29         sstate2new[i,:] += vij
30         sstate2new[j,:] += vij
31
32         #update state2
33         vi=varij*state[:,i]
34         vj=varij*state[:,j]
35         state2new[:,i] += vj
36         state2new[:,j] += vi
37         state2new[i,:] += vj
38         state2new[j,:] += vi
39         state2new[i,i]+=1
40         state2new[j,j]+=1
41
42         deltatwos = -sigma*(np.count_nonzero(sstate2[i,:]==2)
43 - np.count_nonzero(sstate2new[i,:]==2)
44 + np.count_nonzero(sstate2[j,:]==2)
45 - np.count_nonzero(sstate2new[j,:]==2)
46 -(np.count_nonzero(sstate2[j,i]==2)-np.count_nonzero(sstate2new[j,i]==2)))
47
48         delta = 2*(varij*(alpha+beta*(deg[i]+deg[j]))+beta)
49
50         if (np.random.random() < min(1,np.exp(deltatwos+delta))):
51             deg[i]+=varij
52             deg[j]+=varij
53             state2[:,i] = state2new[:,i]
54             state2[:,j] = state2new[:,j]
55             state2[i,:] = state2new[i,:]
56             state2[j,:] = state2new[j,:]
57             sstate2[:,i] = sstate2new[:,i]
58             sstate2[:,j] = sstate2new[:,j]
59             sstate2[i,:] = sstate2new[i,:]
60             sstate2[j,:] = sstate2new[j,:]
61             state[j,i] = 1-state[j,i]
62             state[i,j] = state[j,i]
63
64         else:
65             state2new[:,i] = state2[:,i]

```

```

66     state2new[:,j] = state2[:,j]
67     state2new[i,:] = state2[i,:]
68     state2new[j,:] = state2[j,:]
69     sstate2new[:,i] = sstate2[:,i]
70     sstate2new[:,j] = sstate2[:,j]
71     sstate2new[i,:] = sstate2[i,:]
72     sstate2new[j,:] = sstate2[j,:]
73
74     if step % snap ==0 and plot_t:
75         ydata[0].append(np.mean(deg))
76         ydata[1].append(np.var(deg))
77     step -= 1
78
79     if plot_t:
80         plt.plot(ydata[0])
81         plt.plot(ydata[1])
82         plt.plot(ydata[2])
83         plt.title(str(N)+' beta'+str(beta)+' sigma'+str(sigma))
84         plt.savefig('tri_sweep_report/'+os.path.basename(__file__)[:-3]+'%db%ds%d'%(N
,beta, sigma)+'.png')
85         plt.show()
86         np.savetxt('tri_sweep_report/'+os.path.basename(__file__)[:-3]+'%db%ds%d.txt'
%(N,beta, sigma),ydata)
87
88     return alpha, state, state2, sstate2
89 #####
90 def save_adj(N, alpha, beta, sigma, step, p, sample):
91     state = ini_ER(N, p)
92     state2 = np.matmul(state, state)
93     sstate2 = state2*state
94
95     t = time.localtime()
96     current_time = time.strftime("%H:%M:%S", t)
97     print(current_time)
98
99     k_0 = np.sum(state,axis=0)
100    print(step, alpha, np.mean(k_0), np.var(k_0))
101
102    alpha, state, state2, sstate2 = flip(step, state, state2, sstate2, N, alpha, beta,
sigma)
103
104    t = time.localtime()
105    current_time = time.strftime("%H:%M:%S", t)
106    print(current_time)
107    k_0 = np.sum(state,axis=0)
108    print(alpha, beta, sigma, np.mean(k_0), np.var(k_0))
109
110    N = 200
111    alpha = 177
112    beta = -20
113    sigma = 100
114    save_adj(N, alpha, beta, sigma, 5e5, 6/N, 1000)

```

For the modified square model, we only replace the variation part in the flip

function with the following code:

```

1  varij=1-2*state[i,j]
2
3  #update state2
4  vi=state[:,i]*varij
5  vj=state[:,j]*varij
6  state2new[:,i] += vj
7  state2new[:,j] += vi
8  state2new[i,:] += vj
9  state2new[j,:] += vi
10 state2new[i,i]+=1
11 state2new[j,j]+=1
12
13 delta = 2*(varij*(alpha+beta*(deg[i]+deg[j]))+beta)
14
15 deltatwos = -sigma*(np.count_nonzero(state2[i,:]==2)
16 - np.count_nonzero(state2new[i,:]==2)
17 + np.count_nonzero(state2[j,:]==2)
18 - np.count_nonzero(state2new[j,:]==2)
19 -(np.count_nonzero(state2[j,i]==2)-np.count_nonzero(state2new[j,i]==2)))
20
21
22 if (np.random.random() < min(1,np.exp(deltatwos+delta))):
23     # cube_before = count_cube(state)
24     state[j,i] = 1-state[j,i]
25     state[i,j] = state[j,i]
26     deg[i]+=varij
27     deg[j]+=varij
28     state2[:,i] = state2new[:,i]
29     state2[:,j] = state2new[:,j]
30     state2[i,:] = state2new[i,:]
31     state2[j,:] = state2new[j,:]
32 else:
33     state2new[:,i] = state2[:,i]
34     state2new[:,j] = state2[:,j]
35     state2new[i,:] = state2[i,:]
36     state2new[j,:] = state2[j,:]

```

Bibliography

- [1] Alessia Annibale and Owen T Courtney. *The two-star model: exact solution in the sparse regime and condensation transition*. Apr. 2015. URL: <https://arxiv.org/abs/1504.06458>.
- [2] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. *Gephi: An Open Source Software for Exploring and Manipulating Networks*. 2009. URL: <http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154>.
- [3] C Kelly and C A Trugenberger. “Combinatorial Quantum Gravity: Emergence of Geometric Space from Random Graphs”. In: *Journal of Physics: Conference Series* 1275 (Sept. 2019), p. 012016. DOI: [10.1088/1742-6596/1275/1/012016](https://doi.org/10.1088/1742-6596/1275/1/012016). URL: <https://doi.org/10.1088/1742-6596/1275/1/012016>.
- [4] C. Kelly and C. A. Trugenberger. *Combinatorial Quantum Gravity: Emergence of Geometric Space from Random Graphs*. Nov. 2018. URL: <https://arxiv.org/abs/1811.12905>.
- [5] M. E. J. Newman and G. T. Barkema. *Monte Carlo methods in statistical physics*. Clarendon Press, 2011.
- [6] Mark E.J Newman. *Networks*. Oxford University Press, 2019.
- [7] Juyong Park and M. E. J. Newman. “Solution for the properties of a clustered network”. In: *Phys. Rev. E* 72 (2 Aug. 2005), p. 026136. DOI: [10.1103/PhysRevE.72.026136](https://link.aps.org/doi/10.1103/PhysRevE.72.026136). URL: <https://link.aps.org/doi/10.1103/PhysRevE.72.026136>.
- [8] Juyong Park and M. E. J. Newman. “Solution of the two-star model of a network”. In: *Phys. Rev. E* 70 (6 Dec. 2004), p. 066146. DOI: [10.1103/PhysRevE.70.066146](https://doi.org/10.1103/PhysRevE.70.066146).
- [9] Garry Robins et al. “An introduction to exponential random graph (p^*) models for social networks”. In: *Social Networks* 29.2 (2007). Special Section: Advances in Exponential Random Graph (p^*) Models, pp. 173–191. ISSN: 0378-8733. DOI: <https://doi.org/10.1016/j.socnet.2006.08.002>.
- [10] C. A. Trugenberger. “Combinatorial quantum gravity: geometry from random bits”. In: *Journal of High Energy Physics* 2017.9 (2017). DOI: [10.1007/jhep09\(2017\)045](https://doi.org/10.1007/jhep09(2017)045).
- [11] Eric W. Weisstein. *Graph Cycle*. URL: <https://mathworld.wolfram.com/GraphCycle.html>.