Mixing time of random walks on graphs

Min Chen - 陈 敏
Mathematics Department, University of York
Supervisor: Keith Briggs

Complexity research group, BT, Martlesham
mailto:mc190@york.ac.uk, mailto:keith.briggs@bt.com

August 26, 2004

Contents

Chapter 1

Introduction

1.1 Motivation

The theory of random walks has a long history. Since being introduced, it has been applied in physics, economics, biology, traffic networks and internet. People use this model for information exchanging, spreading problems, sampling problems, ranking pages for search engine, network routing and so on. For example, the advent of sensor, wireless ad-hoc and peer-to-peer networks has necessitated the design of distributed, fault-tolerant, computation and information exchange. This is mainly because the topology may not be complete known, or the nodes may join and leave the network so that the network topology itself may change. These constraints motivate the design of simple algorithms for computation where each node exchange information to only its immediate neighbors in each slot of time; this follows the random walk model.

In many applications, we need to know how soon the random walk will become stationary. This relates to how good results we can get after a certain time when we start a random walk, or on what kind of network we can get a fast converging random walk. The classical theory of random walks deals with random walk on simple, but infinite graphs. More recently, people paid more attention to the random walks on more general, but finite graphs. So we can treat a random walk as a finite Markov chain that is time-reversible. In fact, the theory of random walk is quite similar to the theory of Markov chain. This motivates me to analyse the properties of random walks, especially on mixing time, which is the convergence rate of the random walk (or Markov chain).

It is highly likely that the convergence rate of random walks depends on network topology, as the results of our experiments shown. This relates to the second largest eigenvalue of the transition probability matrix, which I will

discuss in more detail later. In order to find out how the topology structures effect the mixing time, I used different graph models in this problem.

There are many theoretical results available for random walks performed on regular lattices such as $\mathbb{Z}^d$. But actually, it has been suggested recently that more complex networks are relevant to the real world. Particularly, the small-world model which was proposed by Watts and Strogatz [WS98] exhibits unusual connection properties and strong clustering with very small average shortest path between two nodes, this is exactly the properties that many complex networks in nature exhibit. M. E. J. Newman mentioned several small-world models built on lattices in [New03], which includes the models I used in this report. These models have been applied to the analysis of various biological, engineering, and social networks including information flow in the Internet. So in this report, I analyse the mixing time of random walks on a general class of random graphs including these special networks to see how topology structures changes convergence rate. More recent research results can be found in the papers [New03], [AB01] and [New02].

1.2 Organization

I begin in the next chapter by reviewing the relevant knowledge of graph theory, including the basic definitions and some important models. In chapter 3, I introduce the random walk and Markov chain, with an example to interpret several important properties of random walk and the formulas to calculate these properties. The fastest mixing problem comes after, which aims to find the smallest mixing time in a set of different networks. Most of the research results are shown in chapter 5. First I explain the theories about the second largest eigenvalue modulus, and list the numerical results of mixing time on small regular graph which is got from our program. Then I construct two small world random graph models by using Bernoulli and Erdős random graph models. Large simulations have been done on these two models to get the mixing time; I focus on the changes on the numerical results while I change the parameters of the models, which means the topology of the networks changes, and derive the relations between the mixing time and topology based on the numerical results. Some illustrative applications of this fastest mixing problem are presented in chapter 5. Finally is the conclusion and future work.

1.3 Acknowledgments

Thanks to BT which gave me the opportunity to join this interesting project. Thanks to my internal supervisor Richard Clegg in York for the reading of this report and for suggesting improvements. And supreme thanks to my supervisor Keith Briggs for insight and guidance at every step of the process.

Chapter 2

Graph Theory

2.1 Some basic definitions

In order to give all the mathematical definitions easily and clearly, it might be best to start with a simple example.

Example: To warm up, consider this example of a path P_5, which has 5 nodes, as the picture shown below:

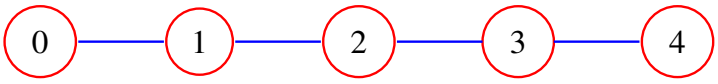


Figure 2.1: 5 nodes path

- ⋆ The *degrees* (number of links) $\deg(i)$ of all the nodes in P_5 are

$$\deg(0) = 1, \quad \deg(1) = 2, \quad \deg(2) = 2, \quad \deg(3) = 2, \quad \deg(4) = 1$$

- ⋆ The *adjacency matrix* A is a $n \times n$ matrix with 1 in A_{ij} if i is connected to j and 0 for the others. $i = 0, 1, \ldots, n-1$, $j = 0, 1, \ldots, n-1$. Here the adjacency matrix for P_5 is:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

⋆ Two other useful matrices for graphs are $\mathscr{L}$ and N. The *normalized Laplacian* $\mathscr{L}$ is the matrix with 1 on the diagonal, $-1/\sqrt{\deg(i)\deg(j)}$ when i is connected to j and 0 at the other places. With the definition of $\mathscr{L}$, N can be easily defined as $N = I - \mathscr{L}$. The matrices of P_5 are shown below.

$$\mathscr{L} = \begin{bmatrix} 1 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ -\frac{1}{\sqrt{2}} & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 \\ 0 & 0 & -\frac{1}{2} & 1 & -\frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 1 \end{bmatrix} \quad N = \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

⋆ *Eigenvalues* of a graph are the eigenvalues of the N matrix. Here are those for graph P_5:

$$\lambda_N = \left\{1, \cos\left(\frac{\pi}{4}\right), \cos\left(\frac{2\pi}{4}\right), \cos\left(\frac{3\pi}{4}\right), \cos(\pi) = -1\right\}$$

After the very intuitive introductions for these basic notions of graph theory, now I present formal mathematical definitions of them. Table 2.1 is a table for all the ideas I used here.

2.2 Important graph models

Here I introduce several different graph models, which I will use in our further research.

1. Deterministic graph models

   First I introduce some deterministic graph models; this is the type of graph which has some special rules on its topological structures. Some normal useful models are list in Table 2.2.

2. Random graph models

   Generally speaking, a random graph is a graph consisting of n vertices and its edge set is generated in some random fashion. However, according to the different ways to generate the edges, we still can divide them into some specific models. There are also many other random graph models, Béla Bollobás introduced more in *Random Graphs* [Bol01]. In Table 2.3 I only list those I use in this report.

   (a) The two most frequently occurring ensembles of random graphs are $\mathscr{G}(n, m)$ and $\mathscr{G}\{n, P(edge) = p\}$ (abbreviated as $\mathscr{G}\{n, p\}$). The definitions and properties are listed in the table below.

graph (simple undirected)	*A graph is an ordered pair of disjoint sets (V, E), which V is a nonempty set of elements, called vertices (or nodes, or points) , E is a list of unordered pairs of the elements in V, which are the edges (or links) of G*
degree	*The degree (or valency) $d_G(v) = \deg(v)$ of a vertex v is the number $\lvert E(v)\rvert$ of edges at v.*
regular	*If all the vertices of G have the same degree k, then G is k-regular, or simply regular.*
walk	*A walk W in a graph is an alternating sequence of vertices and edges, say $x_0, e_1, x_1, e_2, \ldots, e_l, x_l$, where $e_i = x_{i-1}x_i, 0 < i \leqslant l$.*
cycle	*If a walk $W = x_0x_1 \ldots x_l$ is such that $l \geqslant 3$,$x_0 = x_l$, and the vertices x_i, $0 < i < l$, are distinct from each other and x_0, then W is said to be a cycle.*
connected	*A non-empty graph G is called connected if any two of its vertices can be linked by a path in G, and otherwise it is disconnected.*
adjacency matrix	*The adjacency matrix $A = (a_{ij})_{n\times n}$ of G is defined by* $$a_{ij} = \begin{cases} 1 & \text{if } v_iv_j \in E \\ 0 & \text{otherwise} \end{cases}$$
normalized Laplacian	*Let D be the diagonal matrix with D_{ii} the degree of node i, $L \equiv D - A$ be the (combinatorial) Laplacian matrix, the normalized Laplacian, $\mathscr{L} = D^{-1/2}LD^{-1/2}$, is* $$\mathscr{L}(i,j) = \begin{cases} 1, & \text{if } i = j \\ -1/\sqrt{\deg(i)\deg(j)}, & \text{if } i \sim j \\ 0, & \text{otherwise.} \end{cases}$$
N	*$N = I - \mathscr{L}$, which can also be calculated as follow:* $$N(i,j) = \begin{cases} 1/\sqrt{\deg(i)\deg(j)}, & \text{if } i \sim j \\ 0, & \text{otherwise.} \end{cases}$$ where $i \sim j$ means $(i,j) \in E$.
eigenvalues	*The eigenvalues of graph $G(V, E)$ are the eigenvalues $\{\{\lambda_{Ni}\}, i = 1, 2, \ldots, n\}$ of the matrix N.*

Table 2.1: Basic definitions in graph theory

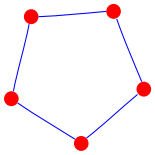
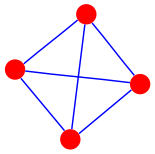
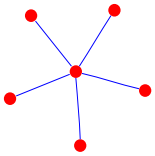
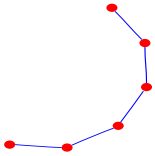
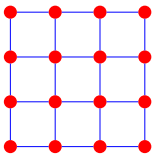
name	example	definition	Symbol
Cycle		*If $P = x_0 \dots x_{k-1}$ is a path and $k \geqslant 3$, then the graph $C := P + x_{k-1}x_0$ is called a cycle.*	C_n
Complete graph		*A graph in which each pair of graph vertices is connected by a graph edge is called complete graph*	K_n
Tree		*A connected graph without any cycles is a tree*	T_n
Path		*A path is a non-empty graph $P = (V, E)$ of the form $V = \{x_0, x_1, \dots, x_k\}$ $E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}$,where the x_i are all distinct.*	P_n
Grid (Mesh)		*Usually a grid refers to two or more infinite sets of evenly-spaced parallel lines at particular angles to each other in a plane, or the intersections of such lines. Mathematically, the $k \times k$ grid is the graph $\{1, \dots, k\}^2$ with the edge set $\{(i, j)(i', j') : \lvert i - i' \rvert + \lvert j - j' \rvert = 1\}$*	G_n

Table 2.2: Several deterministic graph models

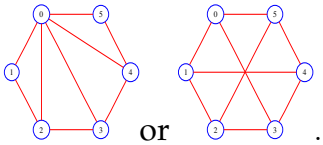
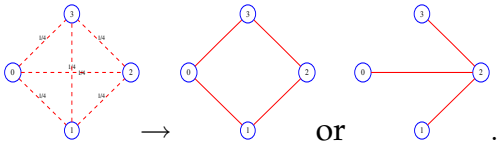
$\mathscr{G}(n,m)$ (Bernoulli)	$\mathscr{G}\{n,p\}$ (Erdős & Rényi)
$\mathscr{G}(n,m)$ is an ensemble of labelled graphs with vertex set $V(G) = 1,2,\ldots,n$, having m randomly chosen edges (where M usually depends on n).	*$\mathscr{G}\{n,p\}$ is an ensemble of labelled graphs with vertex set $V(G) = 1,2,\ldots,n$, in which every one of the possible $\binom{n}{2}$ edges exists with fixed probability $0 \leqslant p \leqslant 1$, independent of any other edges.*
$\mathscr{G}(6,9)$ 	$\mathscr{G}\{4,1/4\}$ 

Table 2.3: Two basic random graph models

(b) Small world graphs

Many complex networks in nature exhibit two properties that are seemingly at odds. The neighbors of neighbors are very likely to be neighbors, and any two nodes can typically be connected by a relatively short path. Watt and Strogatz referred to this as the small world phenomenon [WS98].

In this report, I use a simple variant of this model. I start with n nodes $\{v_i, i = 1,2,\ldots,n\}$ where $n \geqslant 3$, first connect them one by one to be a cycle. So now the graph has edges $\{v_n v_0, v_i v_{i+1}, i = 0,1,\ldots,n-1\}$. Then I add several cross links $\{v_p v_q, 0 \leqslant p \leqslant n, 0 \leqslant q \leqslant n\}$ into the cycle. These graphs have the properties of the small world model.

Chapter 3

Random walks on graphs

3.1 Markov chain

The Markov chain is a widely used model in networks. It is a memoryless process which characterized by a transition matrix P. And it is also an important model for random walks on graphs, to which I will refer now.

For example, consider a connected graph $G(V, E)$ with 4 nodes ($n = 4$) and 4 edges ($m = 4$). V is the vertex set and E is the edge set. We define a discrete-time Markov chain on the vertices of the graph as follows. Denote the state of the graph at each step of time t as X_t, where $t = 0, 1, \ldots$. For each step, X has a transition probability, which is associated to each edge in the graph, to make a transition between the two adjacent vertices. These probabilities are all nonnegative and the sum of the probabilities of edges connected to each vertex (including the self-loop if it exists) must be one.

Usually we describe the Markov chain via the *transition probability matrix* P, which is defined as:

Definition 3.1 *If at time t the state is at node i, then at time $t + 1$ the probability of the move to node j is*

$$P_{ij} = \mathbf{P}(X(t + 1) = j | X(t) = i), \quad i, j = 0, \ldots, n - 1.$$

P is called the transition matrix, which satisfies

$$0 \leqslant P_{ij} \leqslant 1, \quad i, j = 0, \ldots, n - 1; \quad \sum_{j=0}^{n-1} P_{ij} = 1, \quad i = 0, 1, \ldots, n - 1.$$

The definition above implies that P must satisfy

$$P_{ij} = 0, \quad (i,j) \notin E,$$

which states that transitions are allowed only between vertices that linked by an edge.

Let $\pi(t) \in \mathbb{R}^n$ be the probability distribution of the state at time t:

$$\pi_i(t) = \mathbf{P}(X(t) = i).$$

The distribution $\pi(t+1)$ at time $t+1$ can be calculated as:

$$\pi(t+1) = P^T \pi(t)$$

and it is easy to get the general expression for the distribution at any time t via $\pi(0)$ which is the initial distribution at $t = 0$.

$$\pi(t) = (P^T)^t \pi(0)$$

What will π tend to as $t \to \infty$? There is another important definition about Markov chain.

Definition 3.2 *The distribution π is called the equilibrium distribution (or stationary/invariant distribution) if it satisfies:*

$$\pi = \pi P$$

With this definition, we can easily get that for every graph G, the distribution

$$\pi_i = \frac{\deg(i)}{2m} \tag{3.1}$$

is stationary, where $\deg(i)$ is the degree of node i and m is the total number of edges in G. But not all Markov chains have a unique equilibrium distribution. Only when the chain satisfies some restrictions, the equilibrium distribution will exist and be unique. Before we come to these restrictions, first let us have a look at some related definitions.

Definition 3.3 *The n-step transition matrix $P_n = (p_{ij}(n))$ is the matrix of n-step transition probabilities*

$$p_{ij}(n) = \mathbf{P}(X_{m+n} = j | X_m = i).$$

Definition 3.4 *A chain is irreducible if any state can reached from any other, that is for all i and j:*

$$\exists n \ : \ P_{ij}^n > 0$$

Definition 3.5 *State i is called recurrent if*

$$\mathbf{P}(X_n = i \text{ for some } n \geqslant 1 | X_0 = i) = 1,$$

which is to say that the probability of eventual return to i, having started from i, is 1.

Definition 3.6 *The mean recurrence time μ_i of a state i is defined as*

$$\mu_i = \begin{cases} \sum_n n f_{ii}(n) & \text{if } i \text{ is recurrent} \\ \infty & \text{if } i \text{ is transient.} \end{cases}$$

f_{ii} is the first passage times of the chain, defined as

$$f_{ij} = \sum_{n=1}^{\infty} f_{ij}(n)$$

where

$$f_{ij}(n) = \mathbf{P}(X_1 \neq j, X_2 \neq j, \ldots, X_{n-1} \neq j, X_n = j | X_0 = i)$$

μ_i may be infinity even if i is recurrent.

Definition 3.7 *The recurrent state i is called* $\begin{cases} \text{null} & \text{if } \mu_i = \infty \\ \text{non-null (or positive)} & \text{if } \mu_i < \infty. \end{cases}$

Now we can state the theorem about the unique stationary distribution.

Theorem 3.1 *An irreducible chain has a stationary distribution π if and only if all the states are non-null recurrent; in this case, π is the unique stationary distribution and is given $\pi = \mu_i^{-1}$ for each i∈S, where μ_i is the mean recurrence time of i.*

It is easy to see that the states of Markov chain always depends on time t; let us think about the reversal of the time scale of a Markov chain.

Definition 3.8 *Suppose that X_n:$-\infty < n < \infty$ is an irreducible non-null recurrent Markov chain, with transition matrix P and unique stationary distribution π. Suppose further that X_n has distribution π for every $n \in (-\infty, \infty)$. Define the reversed chain Y as:*

$$Y_n = X_{-n}, \quad -\infty < n < \infty$$

It is not difficult to show that Y is a Markov chain also, and of course Y_n has distribution π for each n.

Definition 3.9 *A Markov chain X is called time-reversible if the transition matrices of X and Y are the same.*

Theorem 3.2 *A Markov chain X is time-reversible if and only if* [GS92]

$$\pi_i p_{ij} = \pi_j p_{ji} \quad i, j \in V$$

Proof: The transition probability of Y are (where $m = -n$)

$$\begin{aligned} p_{ij} &= \mathbf{P}(Y_{n+1} = j | Y_n = i) \\ &= \mathbf{P}(X_{-n-1} = j | X_{-n} = i) \\ &= \mathbf{P}(X_m = i | X_{m-1} = j)\mathbf{P}(X_{m-1} = j)/\mathbf{P}(X_m = i) \\ &= p_{ji}\frac{\pi_j}{\pi_i} \end{aligned}$$

by the identity

$$\mathbf{P}(A|B) = \mathbf{P}(B|A)\mathbf{P}(A)/\mathbf{P}(B).$$

Thus $p_{ij} = q_{ij}$ if and only if $\pi_i p_{ij} = \pi_j p_{ji}$

3.2 Random walk

Given a graph and a starting node x_0, we select a neighbor x_1 of x_0 randomly, and move to this neighbor x_1; then we select a neighbor of x_1 randomly, say x_2, and move to it, and so on. The sequence of these randomly selected nodes x_0 is called a *random walk* on the graph.

There is not much difference between the theory of random walks on graphs and the theory of finite Markov chains; every Markov chain can be viewed as a random walk on a directed graph, if we allow weighted edges. Symmetric Markov chains can be viewed as random walks on regular symmetric graphs and time-reversible Markov chains can be viewed as random walks on undirected graphs. Let's see in more detail. Here I consider a simple random walk which chooses the next node from the uniform distribution.

Definition 3.10 *Let $G = (V, E)$ be a connected graph with n nodes and m edges. We call a walk $x_0 x_1, \ldots, x_n$ on G a random walk if at the ith step ($0 \leqslant i < n$), we choose a neighbor of v_i with probability $1/\deg(v_i)$ as the next node, where $1/\deg(v_i)$ is the degree of node v_i.*

Obviously, this random walk is a finite Markov chain. The transition probability matrix P can be calculated as:

$$P_{ij} = \begin{cases} 1/\deg(i), & \text{if } ij \in E. \\ 0, & \text{otherwise.} \end{cases} \tag{3.2}$$

where $\deg(i)$ is the degree of node i.

If we denote the distribution at ith step as $\pi(i)$, the rule of the walk can be expressed by the simple equation:

$$\pi(t) = (P^T)^t \pi(0).$$

It follows that the probability $p_{ij}(t)$ that, starting at node i, we reach node j in t steps is given by ij entry of the matrix P^t.

From Equation 3.2 and 3.1, we get $\pi(i)p_{ij} = 1/(2m)$, where $i, j \in V$ and m is the number of edges in G. So we move along every edge, in every possible direction with the same frequency. If we are sitting at a node i, the expected number of steps before it returns is $1/\pi(i) = 2m/d(i)$. If G is regular, then the return time is just n, which is the number of nodes.

3.3 Measures of random walk

I now introduce the measures of a random walk that plays the most important role in our paper. First we take an interesting example.

Example: The picture below is the London underground map. Now I pick a part of the map as an example, as shown in Figure 3.2. Consider starting at any one stop, and choose which one to go next randomly, that is a random walk on this graph. Now I am going to discuss some interesting questions.

1. If we start a random walk at King's Cross, what is the expected number of steps before we reach to Victoria?

   Here I introduce a notion: *hitting time* $H_{i,j}$, which is the expected number of steps before node j is visited in a random walk starting at i. We can work out the value of $H_{i,j}$ by some certain formula which I will introduce later. In our example, the hitting time from King's Cross to Victoria is $H_{8,1} = 38.5$. Actually, we can easily calculate the hitting time $H_{i,j}$ for any i,j in a connected graph. If we put them in a matrix, say H, we can get the hitting time of any two nodes in the graph from H. I labelled all the nodes in the map above with $0, 1, \ldots, 18$.

   This is the hitting time matrix:

   With this matrix, contrast Figure 3.2 and Figure 3.3, we can get the hitting time for any two tube stops in the map. For example, hitting time from Monument to Baker Street is $H_{4,9} = 31.4$, which means if we start a random walk from Monument, we will visit about 31.4 the other stops before we reach Baker Street.

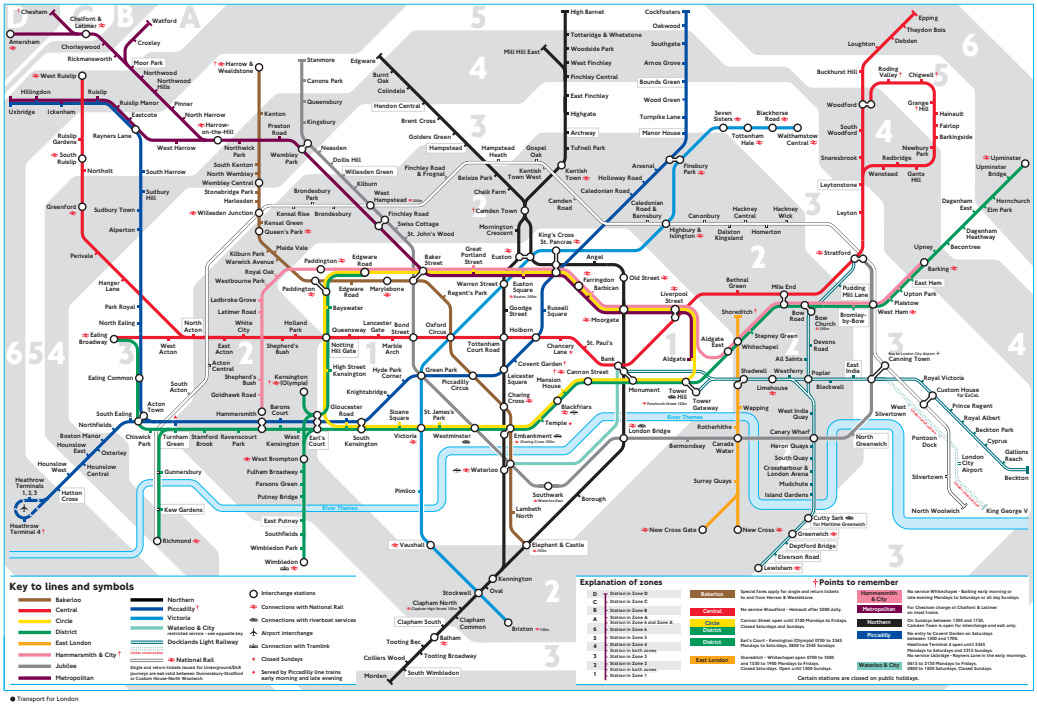


Figure 3.1: London tube map

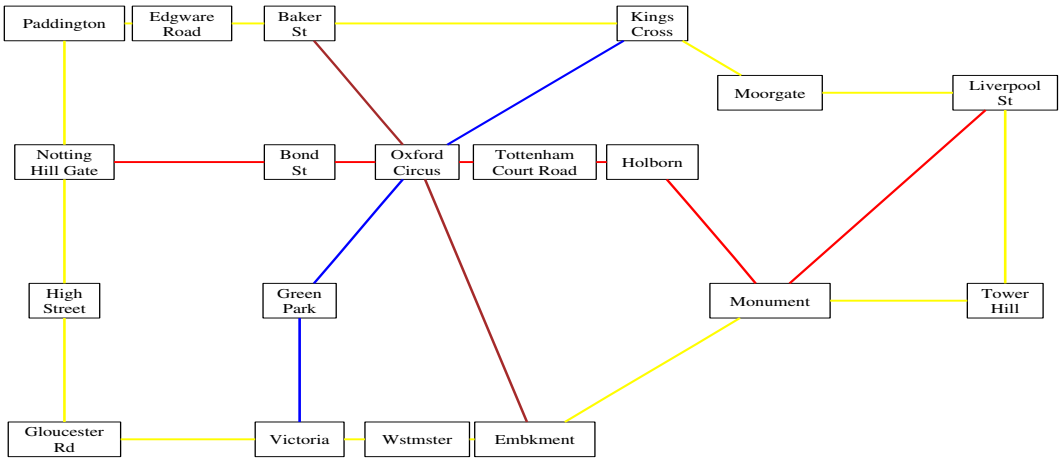


Figure 3.2: Part of London tube map

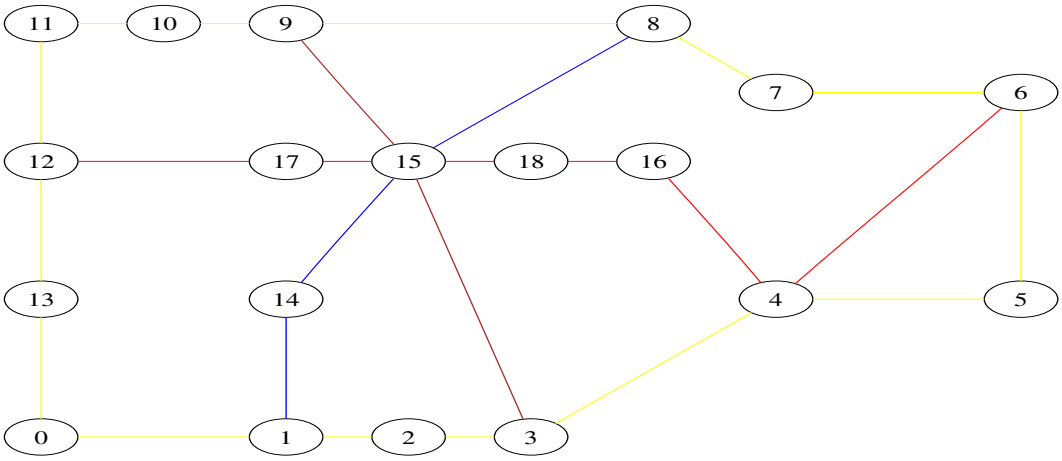


Figure 3.3: Labelled tube map

2. Now let us think about what is the expect number of steps if we start at King's Cross and back to it again via Victoria.

   Here refer to another notion: *commute time* $k(i, j)$. This is the expected number of steps in a random walk starting at i and the first time it return to i via j. It can be calculated by:

   $$k(i, j) = H_{i,j} + H_{j,i}$$

   So we get that the commute time return to King's Cross via Victoria is:

   $$k(8, 1) = H_{8,1} + H_{1,8} = 38.5 + 29.6 = 68.1 \quad .$$

3. We know that a Markov chain will converge to its stationary distribution if it is ergodic. So here if we start a random walk on the tube map, for each step the probabilities of all the stops that we may at are always changing until the chain reaches to the equilibrium distribution. After that, these probabilities are fixed and will not change any more. The question is, how fast it will converge to the stationary distribution for a random walk? And how many steps it will take?

   The answer should be the *mixing rate* and *mixing time*. These are the most important properties I studied. Mixing rate is a measure of how fast the random walk converges to its limiting distribution. It can be calculated by the eigenvalues of the transition probability matrix P; I will discuss it in more detail later. Mixing time can describe the convergence rate more intuitively. It equals $1/$(mixing rate), which means the number of steps before the distribution of a random walk be stationary. In our example, the mixing rate and mixing time are:

   $$\text{mixing rate} = 0.101568$$

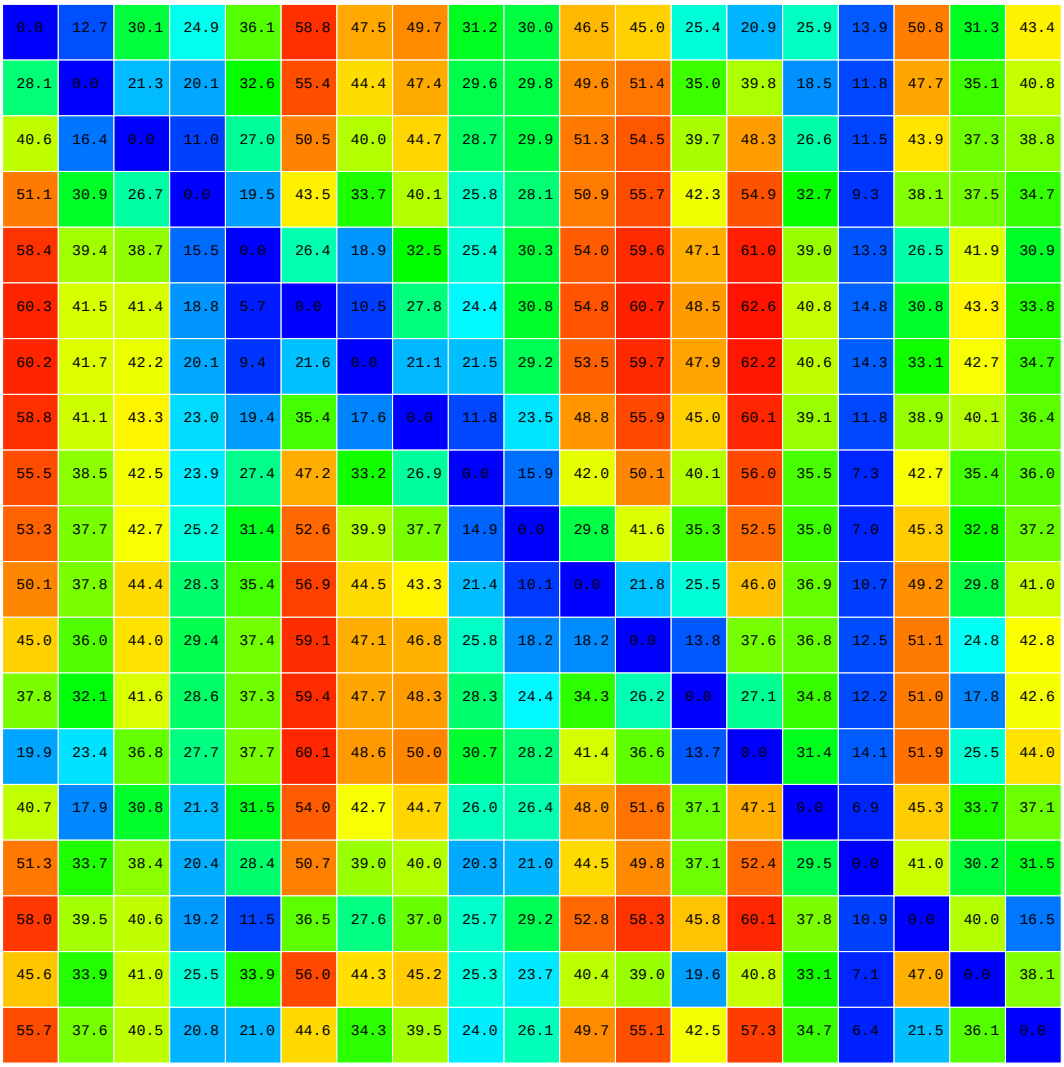


Figure 3.4: Hitting time matrix

$$\text{mixing time} = 9.845655$$

To make the result easier to understand, I add some more edges to the Figure 3.2. Now the tube map looks like:

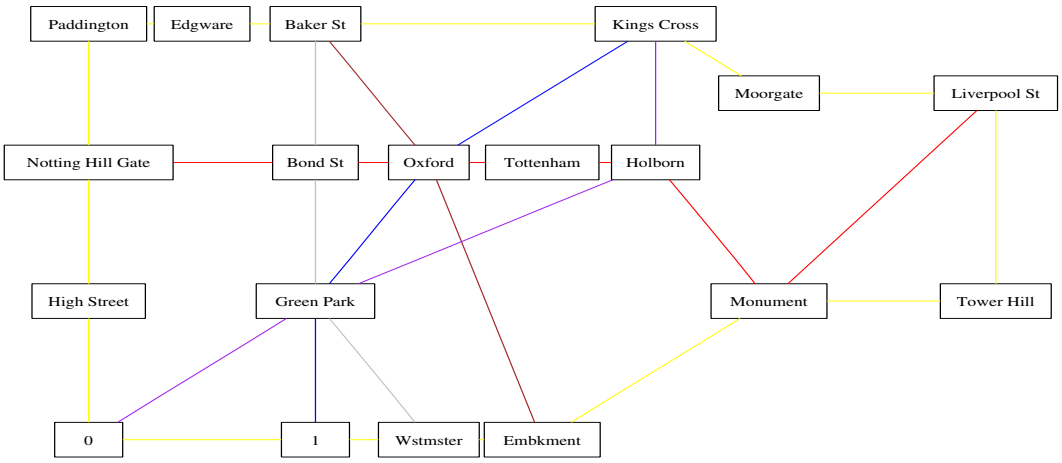


Figure 3.5: Labelled tube map

I calculated the mixing rate and mixing time again, this time we get:

$$\text{mixing rate} = 0.168313$$

$$\text{mixing time} = 5.941294$$

Notice that the mixing rate increased and mixing time decreased. It shows that the graph has a faster convergence rate after I add some more cross links into it.

Table 3.1 is the interpretations of these measures in random walks.

We defined the definitions of the main parameters in random walks, but how can we get these values, how to calculate them? Here I am going to introduce the methods now.

Recall that the probability p_{ij}^t that the random walk starting at i and will be at j after t steps, is the ij entry of P^t. This suggests that we can use the spectral theory of matrices to solve these problems.

The matrix P has largest eigenvalue 1, with corresponding left eigenvector π and right eigenvector $\mathbf{1}$, the all-1 vector on V. In fact, $\pi P = \pi$ express that π is the stationary distribution, while $P\mathbf{1} = \mathbf{1}$ means that exactly one step is made from each other.

The hitting time is determined by the eigenstructure of transition probability matrix P. Here P is not symmetric unless the graph is regular, but it is easy to make it into a symmetric form.

hitting time (access time)	*in a random walk starting from node i, hitting time $H_{i,j}$ is the expected number of steps before node j.*
commute time	*commute time $k(i,j)$ is the expected number of steps in a walk starting at i for first return to i via j.*
cover time	*the expected number of steps to reach every node.*
mixing rate	*measure of how fast the random walk converges to its stationary distribution.*
mixing time	*measure of the number of steps for the distribution to reach the stationary distribution.*

Table 3.1: Interpretation for properties of random walk

Consider a graph $G(V,E)$ with adjacency matrix A and transition probability matrix P. We have the relation $P = DA$, where D is the diagonal matrix $D = \text{diag}(1/\deg(i))$ which I also used in section 2.1. Now think about the matrix $N = D^{1/2}AD^{1/2} = D^{-1/2}MD^{1/2}$. This is symmetric and has the same eigenvalues with P. Here's the proof.

Proof: since

$$N = D^{1/2}AD^{1/2}$$

we can get

$$A = D^{-1/2}ND^{-1/2} \tag{3.3}$$

substitute equation 3.3 into P, we get

$$P = DA = DD^{-1/2}ND^{-1/2} = D^{1/2}ND^{-1/2} \tag{3.4}$$

suppose

$$Pv = \lambda v \tag{3.5}$$

then use equation 3.4 into equation 3.5

$$D^{1/2}ND^{-1/2}v = \lambda v$$

$$D^{-1/2}D^{1/2}ND^{-1/2}v = D^{-1/2}\lambda v$$

$$ND^{-1/2}v = \lambda D^{-1/2}v$$

let

$$w = D^{-1/2}v$$

so λ is the eigenvalue of N, with the eigenvector w

Because N is symmetric, the eigenvalues of N are real, and by Perron-Frobenius theory, are all less than one in modular. Write them in an non-increasing order:

$$1 = \lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n \geqslant -1$$

The method to calculate hitting time is from a theorem.

Theorem 3.3 *Hitting time $H_{i,j}$ can be calculated as:*

$$H_{ij} = 2m \sum_{k=2}^{n} \frac{1}{1-\lambda_k} \left(\frac{v_{ki}^2}{\deg(i)} - \frac{v_{ki}v_{kj}}{\sqrt{\deg(i)\deg(j)}} \right)$$

where λ_k is the kth eigenvalue of N, v_{ki} is the ith component of the normalized eigenvector corresponding to λ_k, $\deg(i)$ is the degree of node i and m is the number of edges in the graph.

L. Lovász describes the deduction of the formula above in [Lov93], which I am not going to go through here.

We can easily calculate the commute time by hitting time, since they have the relation $k(i,j) = H_{i,j} + H_{j,i}$.

The *mixing rate* can be achieved as follows. If the graph is non-bipartite (A graph $G = (V, E)$ is bipartite if V admits a partition into 2 classes such that every edge has its ends in different classes. Clearly, a bipartite graph cannot contain an $odd\ cycle$, a cycle of odd length. The bipartite graphs are all periodic, with periods 2, so they are not ergodic and do not have a unique equilibrium distribution), then $p_{ij}^t \to d_j/(2m)$ as $t \to \infty$, and the mixing rate is

$$\mu = \limsup_{t\to\infty} \max_{i,j} \ |P_{ij}(t) - d(j)/(2m)|^{1/t}$$

And the *mixing time* is the reciprocal of *mixing rate*

$$\text{mixing time} = \tau = 1/\log(1/\mu)$$

3.4 Fastest mixing problems

The rapidly mixing Markov chain problem has been much studied. We are looking for the fastest mixing time from all the graphs, so I focus on the different mixing time of graphs with different topology. Stephen Boyd did some research on the fastest mixing problem too, but changed the weights on each edge and fixed the graph structure (in fact, he also changed the structure since he changed the weights by adding self-loops to each node).

Here's a table which compares the two different method for finding the fastest mixing Markov chain. ($P_{\bullet} \geqslant 0$ means all the components in P are greater than or equal to 0)

Boyd – continuous	This work – discrete
find the fastest mixing Markov chains from all the chains with different weights	find the fastest mixing Markov chains from all the chains with different topology structures
fixed the topology structures of graphs, change the transition probability p_{ij} in P by adding self-loops to the nodes, find a optimal P which gives the fastest mixing time	fix P as $1/\deg(i)$ for p_{ij}, find a optimal graph which has the fastest mixing time by changing the degree $\deg(i)$ of the node
$\text{minimize}_P \quad \mu(P)$ subject to $\quad P_{\bullet} \geqslant 0$ $P\mathbf{1} = \mathbf{1}$ $P^T = P$ $P_{i,j} = 0, \quad i,j \notin E$	$\text{minimize}_G \quad \mu(P(G))$ subject to $\quad P_{\bullet} \geqslant 0$ $P\mathbf{1} = \mathbf{1}$ $P_{ij} = \begin{cases} 1/\deg(i), & \text{if } ij \in E \\ 0, & \text{otherwise} \end{cases}$

Table 3.2: Fastest mixing problem

There is a further problem: if we can find the fastest mixing chain both on topology and weights? We combined Boyd's idea and ours together. First we find a set of graphs with all the different topological structures, then find the fastest weight on each different topology. Finally we pick out the fastest chain from the set of the graphs with fastest weights. This chain is the fastest one both on topology and weights. This optimization problem is a semidefinite program, for which I used the `sdpsol` package [WB96].

FMSMC	FMRMC
find the fastest mixing symmetric Markov chains	find the fastest reversible mixing Markov chains
first find the fastest weights on all different topological symmetric chains, then find the fastest mixing one from these fastest weights	first find the fastest weights on all different topological reversible chains, then find the fastest mixing one from these fastest weights
$\min_{P(G)} \min_P \quad \max\{\lambda_2(P), \lambda_n(P)\}$ s.t. $\quad P\mathbf{.} \geqslant 0$ $P\mathbf{1} = \mathbf{1}$ $P^T = P$ $P_{i,j} = 0, \quad i,j \notin E$	$\min_{P(G)} \min_P \quad \max\{\lambda_2(P), \lambda_n(P)\}$ s.t. $\quad P\mathbf{.} \geqslant 0$ $P\mathbf{1} = \mathbf{1}$ $\Pi P = P^T \Pi$ $P_{i,j} = 0, \quad i,j \notin E$ here $\Pi = \text{diag}(\pi_i)$

Table 3.3: Combined fastest mixing problem

Chapter 4

Computational results

4.1 Second largest eigenvalue modulus (SLEM)

In the last section, I discussed about the methods to calculate the mixing rate, which I use the transition probability p_{ij}^t. In fact, the mixing rate is also determined by the eigenstructure of transition probability matrix P as hitting time. Now I am going to talk about another method to calculate the mixing rate which is much easier.

I will refer to *spectral graph theory* here. The use of matrix theory and linear algebra to analyse graphs is called *spectral graph theory*. Actually I use the eigenvalues of our P matrix here instead of the adjacency matrix often used in spectral graph theory; they are closely related to each other. I will show the relations between them in the next section.

Assume we have all the eigenvalues of matrix P $\{\lambda_0, \lambda_1, \ldots, \lambda_{n-1}\}$. We know that the eigenvalues of P are the same of N, so if we write the eigenvalues in the non-increasing order as the same in section 3.3, follow the Perron-Frobenius theory, we will have

$$1 = \lambda_0(P) \geqslant \lambda_1(P) \geqslant \lambda_1(P) \geqslant \cdots \geqslant \lambda_{n-1}(P) \geqslant -1$$

The convergence rate of a random walk/Markov chain to the equilibrium distribution is determined by the second largest eigenvalue modulus (SLEM) of P.

$$\mu(P) = \max_{i=2,\ldots,n} |\lambda_i(P)| = \max\{\lambda_2(P), -\lambda_n(P)\}$$

The smaller the SLEM, the faster the random walk/Markov chain converges to its equilibrium distribution.

Here are some theorems about the SLEM which are deduced from spectral graph theory. [Chu97]

Lemma 4.1 *For a graph G on n vertices, if μ is the second largest eigenvalue of matrix P,*

1. *for $n \geqslant 2$, we have*

$$\begin{cases} \mu \geqslant \frac{1}{n-1} & \text{if } G \text{ is not complete} \\ \mu = \frac{1}{n-1} & \text{if } G \text{ is complete} \end{cases}$$

2. *for all $i \leqslant n-1$, we have*

$$\lambda_i \geqslant -1$$

with $\lambda_{n-1} = -1$ if and only if G is bipartite.

Lemma 4.2 *For a k-regular graph G on n vertices, we have*

$$\mu(P) \geqslant \sqrt{\frac{n-k}{(n-1)k}}$$

Lemma 4.3 *For a graph G on n vertices, let d_H denote the harmonic mean of the $\deg(v)$, i.e.,*

$$\frac{1}{d_H} = \frac{1}{n}\sum_v \frac{1}{\deg(v)}.$$

SLEM of P $\mu(P)$ satisfies

$$1 + (n-1)\mu(P)^2 \geqslant \frac{n}{d_H}(1 - (1+\mu(P))(\frac{\bar{d}}{d_H} - 1)),$$

where $\bar{d}$ denotes the average degree of G.

Lemma 4.4 *For any fixed k and for any infinite family of regular graphs with degree k, we always have*

$$\liminf_{n\to\infty} \mu(P) \geqslant 2\frac{\sqrt{k-1}}{k}$$

Here we required the chains to be reversible, which makes the eigenvalues of P real. It is equivalent to saying that for every pair $i, j \in V$, $\pi(i)p_{ij} = \pi(j)p_{ji}$. This means that in a stationary walk, we step as often from i to j as from j to i. In our cases, we have $\pi(i)p_{ij} = 1/(2m)$ for $ij \in E$, which means the chains do have the property of time-reversibility.

If the Markov chain is ergodic (similar to random walk), which means it has a unique equilibrium distribution, then $\mu(P) < 1$ and the distribution converges to uniform as about $\mu(P)^t$. Now we can have the expression for mixing rate by $\mu(P)$:

$$\text{mixing rate} = \log(1/\mu(P))$$

and the mixing time is the reciprocal of mixing rate

$$\text{mixing time} = \tau = 1/\log(1/\mu)$$

We can always get the mixing time if mixing rate is not equal to 0. But when $\mu(P) = 1$, which means mixing rate $= \log(1) = 0$. So we cannot get mixing time from it. This is exactly the case for the bipartite graphs. They are not ergodic and do not have a unique equilibrium distribution. So they always have $\lambda_n = -1$. That's why we can only talk about mixing time and mixing rate for non-bipartite graphs.

4.2 Related matrices and eigenvalues

SLEM uses the transition probability matrix P to calculate the eigenvalues. I referred to matrix N, which has the relationship $N = D^{1/2}AD^{1/2}$ with adjacency matrix A, has the same eigenvalues of P. So we got

$$\mu = \max\{\lambda_2(P), -\lambda_n(P)\} = \max\{\lambda_2(N), -\lambda_n(N)\}$$

This is used to calculate the hitting time since the eigenvectors of N are different from P.

I also referred to normalized Laplacian matrix L, which is $\mathscr{L} = I - N$. These matrices are all related to the degrees and adjacency matrix A, so we compared the eigenvalues of the four matrices $A,\ \mathscr{L},\ N,\ P$, here I use cycle, complete graph, star and path as four typical cases.

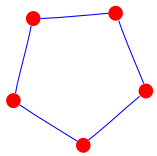
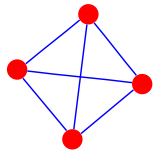
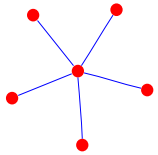
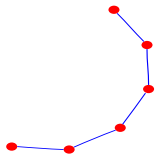
Graph			Eigenvalues $(\lambda_1, \ldots, \lambda_n)$, $(\mu = \max\{\lvert\lambda_2\rvert, \lvert\lambda_n\rvert\})$			
name	example	type	A (Biggs) [Big93]	$\mathscr{L}$ (Chung) [Chu97]	N (Lovász) [Lov93]	P (Lovász) [Lov93]
C_n		Cycle	$2\cos\left(\frac{2k\pi}{n}\right)$, $k = 0, \ldots, n-1$	$1 - \cos\left(\frac{2k\pi}{n}\right)$, $k = 0, \ldots, n-1$	$\cos\left(\frac{2k\pi}{n}\right)$, $k = 0, \ldots, n-1$	The same as N
			$\mu = 2\cos\left(\frac{2\lfloor n/2\rfloor\pi}{n}\right) = \begin{cases} 2\cos\left(\frac{\pi}{n}\right) & n \text{ odd} \\ 2 & n \text{ even} \end{cases}$	$\mu = 1 - \cos\left(\frac{2\lfloor n/2\rfloor\pi}{n}\right) = \begin{cases} 1 - \cos\left(\frac{\pi}{n}\right), & n \text{ odd} \\ 2, & n \text{ even} \end{cases}$	$\mu = \cos\left(\frac{2\lfloor n/2\rfloor\pi}{n}\right) = \begin{cases} \cos\left(\frac{\pi}{n}\right), & n \text{ odd} \\ 1, & n \text{ even} \end{cases}$	
K_n		Complete graph	$(n-1)$, $(-1)_{(n-1)}$	0 , $\frac{n}{n-1}{}_{(n-1)}$	1 , $\frac{-1}{n-1}{}_{(n-1)}$	
			$\mu = 1$	$\mu = \frac{n}{n-1}$	$\mu = \frac{1}{n-1}$	
S_n		Star	$\sqrt{n-1}$, $0_{(n-1)}$, $-\sqrt{n-1}$	0 , $1_{(n-1)}$, 2	-1 , $0_{(n-1)}$, 1	
			$\mu = \sqrt{n-1}$	$\mu = 2$	$\mu = 1$	
P_n		Path	$2\cos\left(\frac{k\pi}{n+1}\right)$, $k = 1, \ldots, n$	$1 - \cos\left(\frac{k\pi}{n-1}\right)$, $k = 0, \ldots, n-1$	$\cos\left(\frac{k\pi}{n-1}\right)$, $k = 0, \ldots, n-1$	
			$\mu = 2\cos\left(\frac{\pi}{n+1}\right)$	$\mu = 1 - \cos\left(\frac{\pi}{n-1}\right)$	$\mu = \cos\left(\frac{\pi}{n-1}\right)$	
Relationships of λ				$\lambda_{\mathscr{L}} = 1 - \lambda_N$	$\lambda_N = 1 - \lambda_{\mathscr{L}}$	
Relationships of matrices			$A = D^{1/2} N D^{1/2}$	$\mathscr{L} = I - N$	$N = D^{1/2} A D^{1/2}$	$P = D^{1/2} N D^{-1/2}$
						$P = DA$

Table 4.1: relationship between eigenvalues of matrices

4.3 Small graphs - regular graphs

I divide the graphs according to the number of nodes. More than 50 nodes are large graphs, 20 to 50 are medium graphs. The graphs with less than 20 nodes I call small graphs. Before we come to the large graphs which can be a model to the network, I did much studies on small graphs first since the number of small graphs with fixed nodes will not be too large, we can do the research over all cases, which will be easier to test ideas and find out some properties. Here I focus on the regular graphs.

SLEM has a constraint: we require a reversible chain to make the eigenvalues real. So I tried small regular graphs, which have the same degree for all nodes. The is the special case of reversible chains, since they always have a symmetric transition probability matrix P. I calculate the mixing rate and mixing time, and find out the minimum and maximum, also the average of all the graph in the same set. And since the bipartite graphs do not have a stationary distribution, I count all the cases out which will get 1 for SLEM. Table 4.2 is part of our result. The complete table is in Appendix A.1.

From the table in Appendix A.1, we can see that the average mixing time is increasing as n increases and decreasing as degree increases. It is easy to think about. If the nodes increases, that means one need to visit more nodes on the random walk, that will take more steps before it reach to the equilibrium state, which means the mixing time will increases. And when the degree increases, we will have more available paths to take to reach the other nodes, in other words, we have the probability to take a short path than a long one. This will make the convergence faster, and reduce the mixing time.

Also I applied the FMRMC algorithm to find the fastest reversible Markov chains (weighted directed graphs). And I list part of the results in Table 4.3. The complete table is in Appendix A.2

By solving the SDP optimization problem, we got all the results in the table in Appendix A.2 for the small regular graphs. Compared to the results shown in Appendix A.1, we can see that some fastest mixing chains are improved, some remain the same. The improved results are in fact directed weighted graphs since we search for the fastest one by changing transition probability matrix P.

4.4 Random large graphs - small world

After some research on small graphs we come to large graphs, which can be up to thousands of nodes and can represent the real networks we have in our life.

Table 4.2: Mixing time on small regular graph (part)

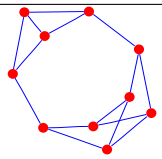
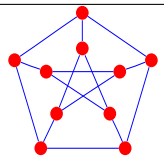
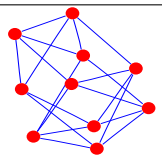
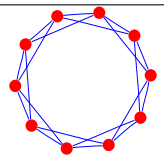
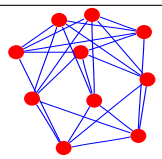
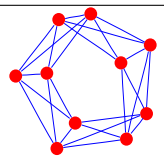
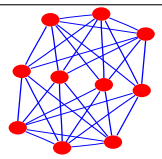
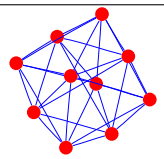
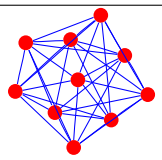
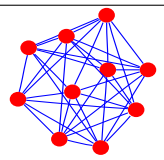
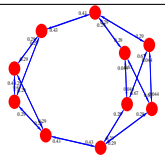
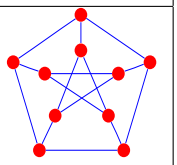
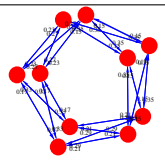
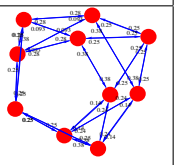
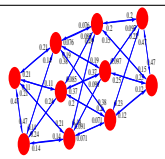
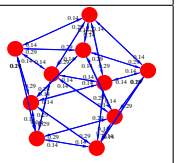
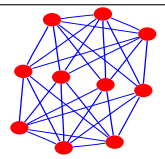
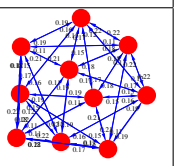
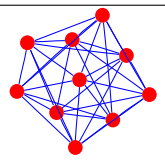
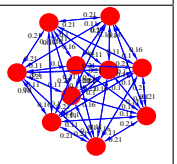
n	deg	num	maxtime	graph	mintime	graph	avertime
10	3	17	15.5896		2.4663		6.8216
10	4	58	7.7220		1.7195		3.4093
10	5	59	7.4542		1.2427		2.2145
10	6	21	2.4663		0.9102		1.5722
10	7	5	1.1802		1.0168		1.1475

Table 4.3: FMRMC on small regular graphs (part)

nodes	degree	max time	graph	min time	graph
10	3	14.9392		2.4663	
10	4	4.7185		1.5767	
10	5	4.3522		1.1802	
10	6	2.4663		0.9064	
10	7	1.1802		0.7670	

I combined the random graph models together to make the problems easier to understand. Here, basically I studied the small world model for random graphs. I applied the two basic models $G(n, m)$ and $G\{n, p\}$ which I referred in former section to both small world and scale free graphs to see how the mixing time changes as the graphs change their topology structure.

1. $SW(n, m)$

   First I applied the $G(n, m)$ model referred in chapter 2 on small world. We call it $SW(n, m)$ and build the whole model as follows:

   (a) Generate a cycle with n nodes, so we have n nodes and n edges in the graph.

   (b) Choose randomly two nodes i,j which are not connected, add an edge between them.

   (c) Keep on adding edges until there are $n + m$ edges in total (which means we add m more edges to the original cycle) .

   Obviously, the mixing time changes as (n, m) changes. I will analyse how these two arguments change the mixing time.

   In the former section, we have already seen that the mixing time for regular graphs are increasing as n increases, and I find that the mixing time is in direct proportion to n^2.

   Let us take the cycle, which is the simplest one, as the example to show this relation. We can write the mixing time τ, which is also the second largest eigenvalue of N, in another way, using the $\cos$ format I list in the table in appendix .0.4

$$\mu = \max\{|\lambda_2|, |\lambda_n|\}) = \left|\cos\left(\frac{(n-1)\pi}{n}\right)\right| = \cos\left(\frac{\pi}{n}\right)$$

   Substitute into τ, we get:

$$\tau(\mu) = \frac{1}{\rho(\mu)} = \frac{-1}{\log(\mu)} = \frac{-1}{\log\cos\left(\frac{\pi}{n}\right)} \tag{4.1}$$

   Equation (4.1) can be expanded as Taylor expansion; we get the result from `maxima`:

$$\tau(n) \approx \frac{2n^2}{\pi^2} - \frac{1}{3} - \frac{\pi^2}{30n^2} - \frac{5\pi^4}{756n^4} + O(n^{-6}) \tag{4.2}$$

in order to make the expression simpler, we square root the left part of equation (4.2) and multiply with the coefficient $\pi/\sqrt{2}$, then from `maxima` we get the Taylor expansion:

$$\sqrt{\frac{\tau}{2}}\pi = n - \frac{\pi^2}{12n} - \frac{17\pi^4}{1440n^3} + \dots$$

for large n, we have

$$\sqrt{\frac{\tau}{2}}\pi \approx n$$

So if we calculate mixing time τ of circles with different number of nodes n, there should be a linear relationship:

$$\sqrt{\frac{\tau}{2}}\pi \approx n \tag{4.3}$$

Now we proved that mixing time is in direct proportion to n^2, which is the same as

$$\sqrt{\frac{\tau}{2}}\pi \sim n.$$

I collected the mixing time τ for graphs with different n and m (since the m more edges are generated randomly, I ran the program for 100 trials to get the average of mixing time for each set of (n, m)), and then separated them into several groups by different values of m. In each set, I find the linear relation on n and $\sqrt{\frac{\tau}{2}}\pi$ by using `linfit` (see appendix 1), and get the result as follows, where $x = n$ and $y = \sqrt{\frac{\tau}{2}}\pi$.

$$\begin{aligned}
0cl \quad & y = -0.108121 + 1.00231x \\
1cl \quad & y = 0.301142 + 1.00392x \\
2cls \quad & y = 1.1288 + 0.739733x \\
3cls \quad & y = 0.754638 + 0.601531x \\
4cls \quad & y = 0.568563 + 0.51182x \\
5cls \quad & y = 1.15241 + 0.431276x \\
6cls \quad & y = 1.1015 + 0.385293x \\
7cls \quad & y = 1.24153 + 0.344286x \\
8cls \quad & y = 1.03032 + 0.320241x \\
9cls \quad & y = 1.19224 + 0.291103x \\
10cls \quad & y = 1.21888 + 0.270236x
\end{aligned}$$

The result shows that n and $\sqrt{\frac{\tau}{2}}\pi$ do have linear relationships to each other, which is also true for all the graphs satisfied this model but not only the cycle cases. And the slopes are decreasing as random links m increases.

Figure 4.1 is the plots of the slopes by m:

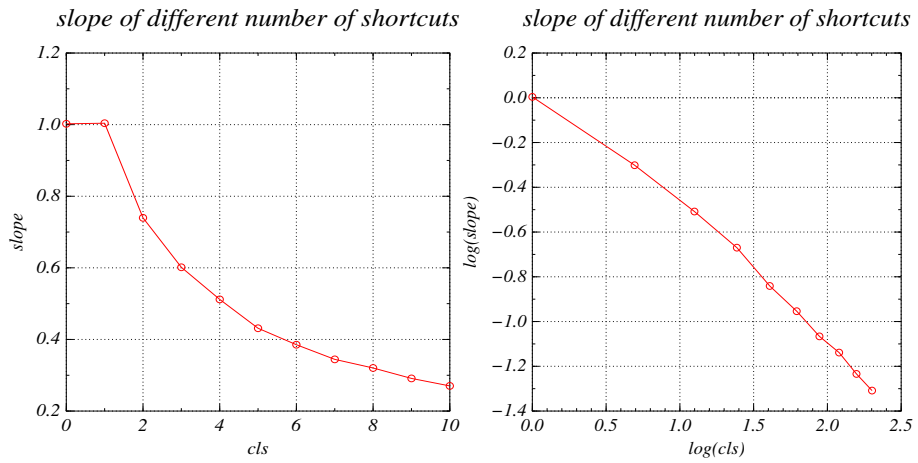


Figure 4.1: Relation between slope($\sqrt{\frac{\tau}{2}}\pi \sim n$) and m

We can see that the slopes are not linearly dependent on m. A tricky thing is that the mixing time for $m = 1$ is surprisingly a bit higher than $m = 0$, which is not in our expectation. I am still working on this try to find out the reason. But in general, we can see the decreasing of slope as m increases very clearly. Although they are not linearly dependent on each other, when I plot them into log scale, the curve became very straight. So it might be that the slope is approximate to m^k.

Now let's see how m affects the mixing time. Figure 4.2 is the plot of $\log$(mixing time) by $\log(m)$ for large graph, the number of the total nodes n is $11, 21, 51, 101, 201, 501, \ldots, 5001$

Since $\log(0)$ cannot be achieved here, so I put the mixing time for $m = 0$ at $\log(nre) = -0.5$ in the picture. The mixing time decreases as the random cross links increases. This is the same results as I did on the small regular graph before. It is quite clear that all these lines are very straight and seem to have similar slopes.

From equation (4.3), we can get

$$\log(\tau) = 2\log(n) + \log(2/\pi)$$

Now I subtract $2\log(n)$ from each data, which makes all the lines start close to the same node $(-0.5, \log(2/\pi))$. Figure 4.3 is the plot:

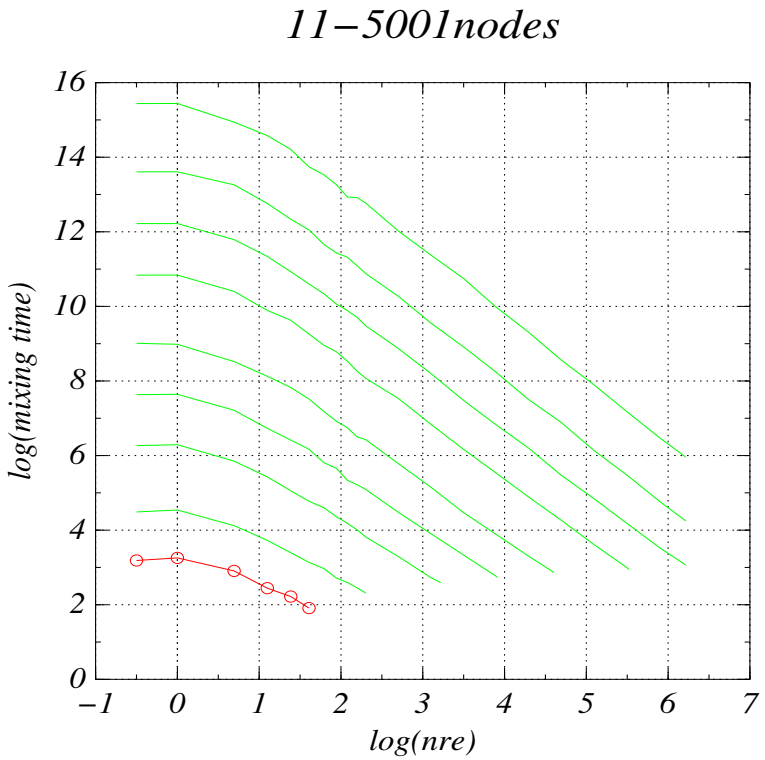


Figure 4.2: Plots of $\log(mixingtime)$ by $\log(m)$ for small world

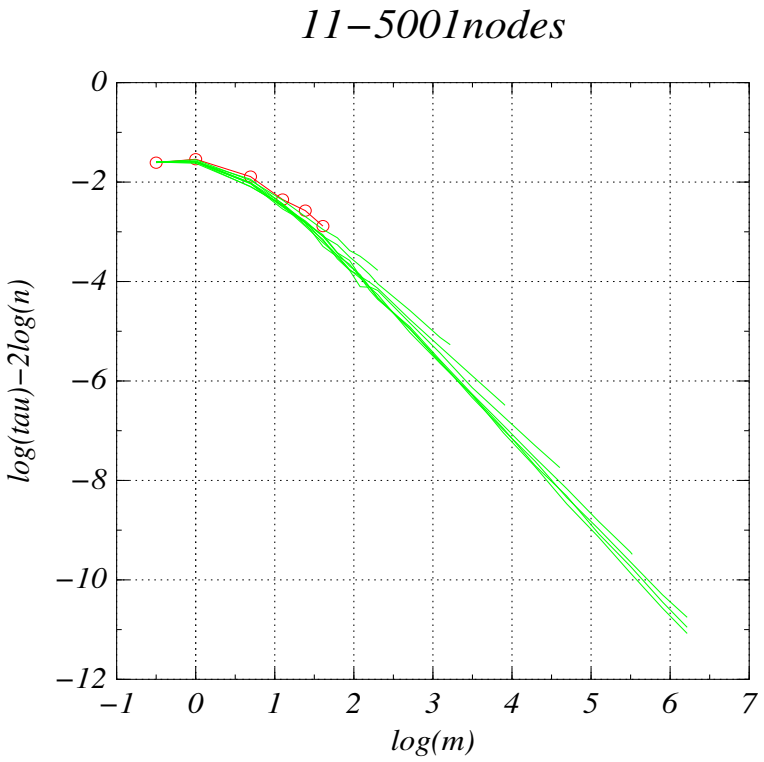


Figure 4.3: Plots of $\log(\mathrm{m}ixingtime)$ against $\log(m)$ start at the same beginning

It is obvious that the mixing time is very large when the graphs have large n and small m. Since the range of mixing time is very large, it is not easy to see how the surface changes step by step, so I made the contour plot for mixing time of $SW(n, m)$ graphs in log scale, which more clearly shows the relations amongst them. Figure 4.4 shows how it looks:

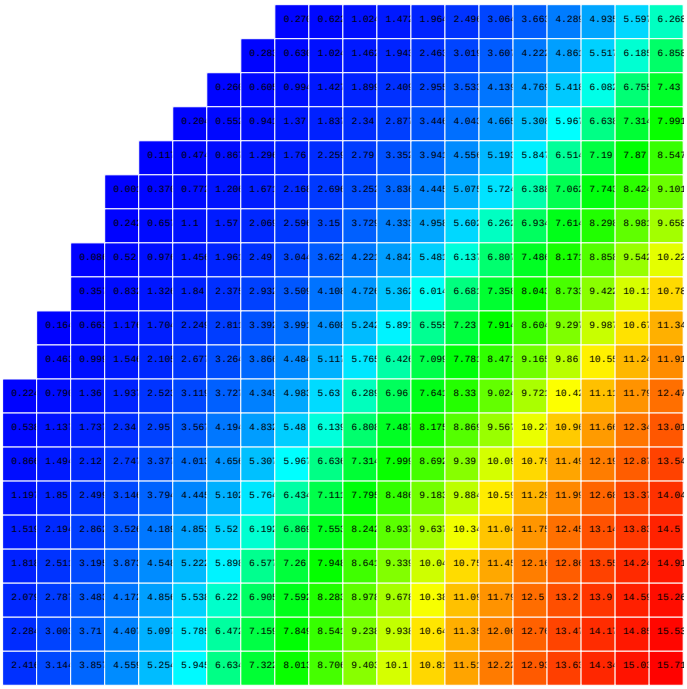


Figure 4.4: Colour plot of mixing time against $x \propto \log(m)$ and $y \propto \log(n)$

The contour plot shows that the mixing time in log scale changes very smoothly as the other two coordinates change, with a certain slope. Also I drew the plot in 3D space. Figure 4.5 is the 3D plot in log scale generated by `maple`.

From this plot, we can see that the surface is almost flat; this also proves that the mixing time is in the directed proportion to n^{k_n} and m^{k_m}, where k_n and k_m are certain numbers:

$$\tau \approx n^{k_n} \approx m^{k_m}.$$

2. $SW\{n, p\}$

   Here we use the model $G\{n, p\}$ on the small world, call it $SW\{n, p\}$. Instead of generating each edge by probability p, I found a more efficient way generate these graphs.

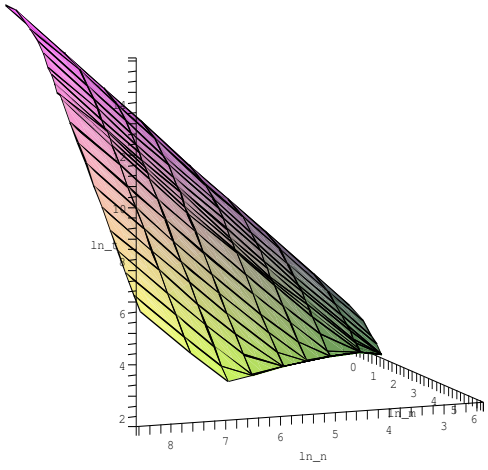


Figure 4.5: 3D Plot of mixing time against n & m in log scale

Suppose we want to generate a set of graphs which satisfy the $SW\{n, p\}$ model, it is easy to see that the probability that a graph has exactly m edges for $0 \leqslant m \leqslant \binom{n}{2}$ fixed is

$$\mathbf{Pr}[\text{m edges}] = \binom{N}{m} p^m (1-p)^{N-m}$$

where $N = \binom{n}{2}$, which means the number of edges in n nodes complete graph. This is exactly binomial distribution. When n become large, $N = \binom{n}{2}$ also become large, this distribution is approximate to Poisson distribution, and m can be calculated from p and n, that is $\bar{m} = Np = \binom{n}{2}p$.

So now we can generate these graphs in a simple way. Similarly we start at n nodes cycle and add some additional cross links to it.

(a) Generate a cycle with n nodes, so we have n nodes and n edges in the graph.

(b) Generate a set of number $m_0.m_1, \ldots, m_k$ which are binomial distributed and the probability $\bar{p}$ of them is fixed.

(c) Generate a set of graph $SW(n, m_0), SW(n, m_1), \ldots, SW(n, m_k)$ using the $SW(n, m)$ model.

I calculated the average mixing time for each group with different mean m, plotting each group with the same $\log(p)$ axis. The pictures in Figure 4.6 are the plots for different number of nodes.

We can see clearly that the $\log(\tau)$ is decreasing, and it is a smooth curve. As n increases, the data become more convergent and the range of mixing time is narrower as p tends to 1. When n is small, $\log(\tau)$ only have a few different values at small p. This because for small n and p, the number of cross-links equals to 1 very often, these different value of $\log(\tau)$ shows all the mixing time with 1 cross-link at different position, which is not many when n is small.

Unfortunately I did not have time to figure out the relations between them. I take 1001 nodes $SW\{n,p\}$, pick out the tail of the curve, which p is approaching to 1. Figure 4.7 is the plot in $\log$ scale:

The plot is very close to straight line. `linfit` shows the linear relation between $\log(\tau)$ and $\log(p)$:

$$y = -1.25033 - 0.31463x$$

So as p tends to 1, the mixing time is approximate to cp^k.

Since the $SW\{n,p\}$ model is based on $SW(n,m)$, for a fixed set (n,p), we can get a mean value of the number of the cross links, which equals to $\binom{n}{2}p$. So with the same n, the values of p in model $SW\{n,p\}$ are proportional to the mean value of m in $SW(n,m)$.

$$p = \frac{2\bar{m}}{n(n-1)}$$

From the result we got for large p:

$$\tau \approx cp^k$$

It can be deduced that

$$\tau \approx \left(\frac{2\bar{m}}{n(n-1)}\right)^k$$

Take $\log$ of both sides:

$$\log(\tau) = k\log(\bar{m}) - k\log(n(n-1)) + k\log(2c)$$

When n is large, $n(n-1) \approx n^2$, we get

$$\log(\tau) \approx k\log(\bar{m}) - 2k\log(n) + C;$$

here C is a constant.

The analysis and results above shows how the mixing time changes by changing the topology of the small world models. In $SW(n,m)$ and $SW\{n,p\}$ when p is large, the mixing time $\tau \approx O(m^{k_m}) \approx O(n^{k_n})$.

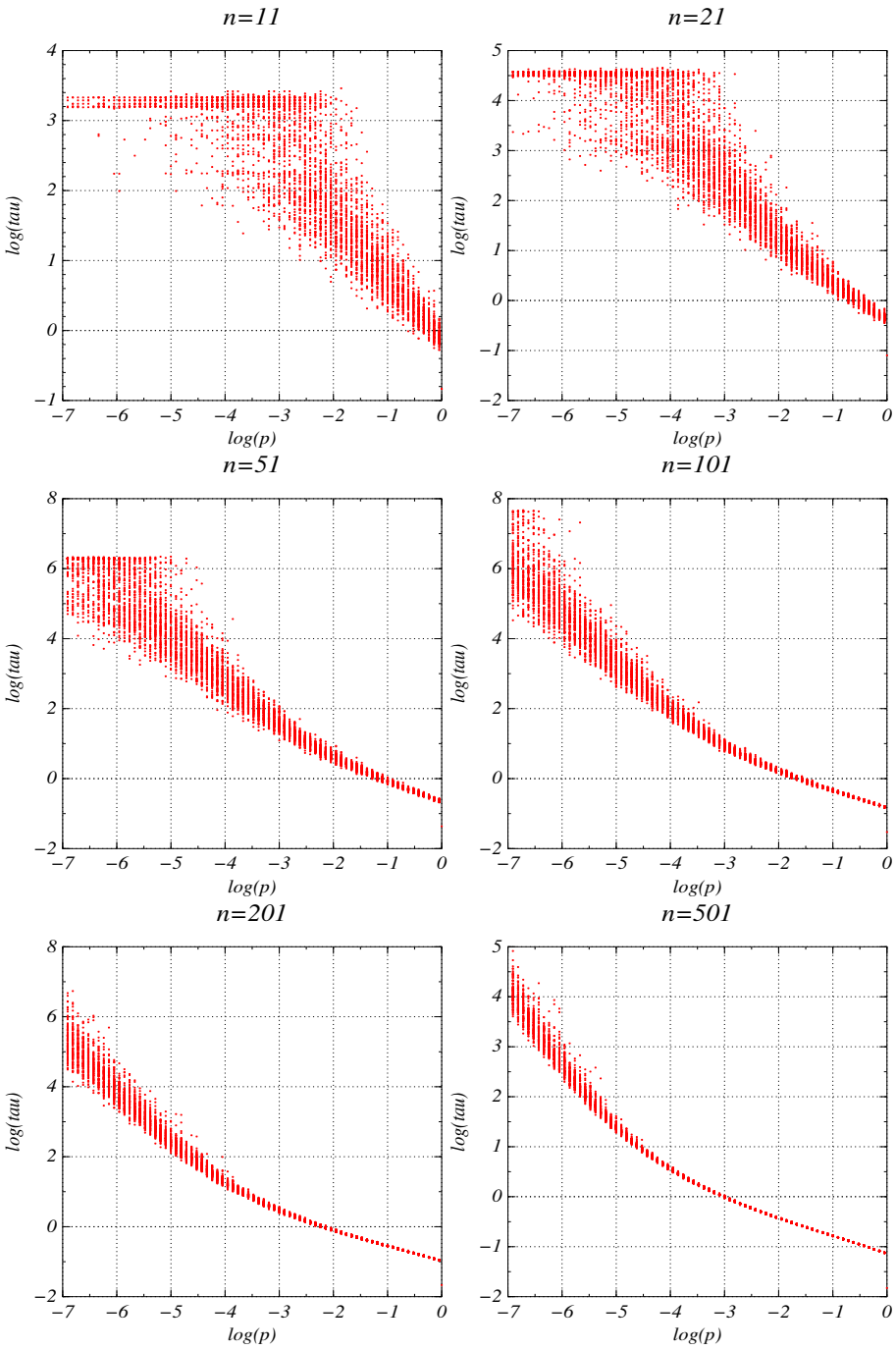


Figure 4.6: Mixing time for $SW\{n, p\}$ graphs for $n = 11, 21, \ldots, 501$ with binomially-distributed cross-link numbers: $m \sim \text{binomial}(\binom{n}{2} - n, p)$

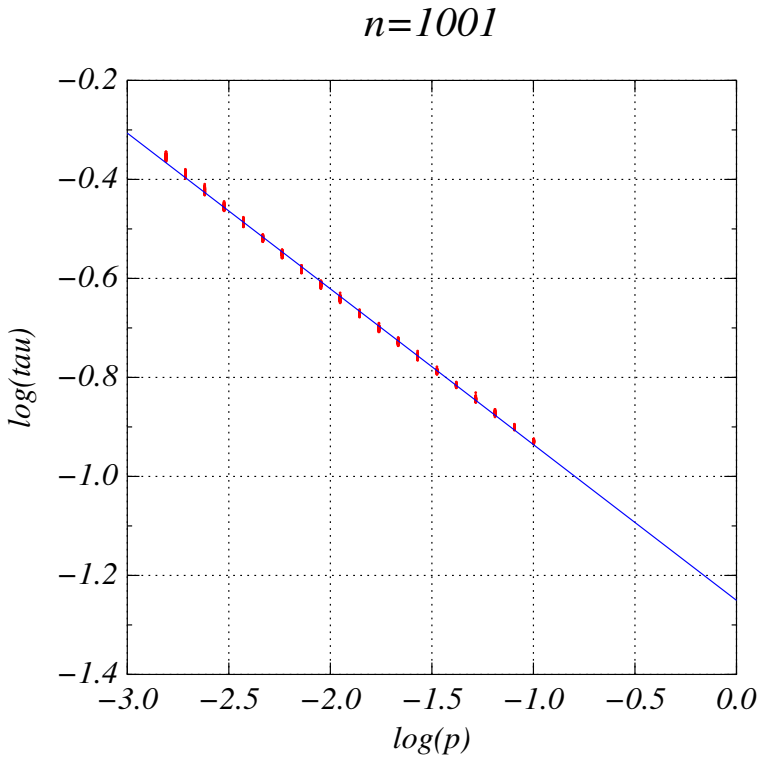


Figure 4.7: Mixing time for $SW\{1001, p\}$ graph with binomially-distributed cross-link numbers: $m \sim$ binomial$(\binom{1001}{2} - 1001, p)$

Chapter 5

Application and conclusion

5.1 Applications

Since random walks have been applied to various fields, the mixing time, as an important property of it, also has many applications. It can be used to measure the random walk on a certain network, or for some random walk applications find the optimized network to get a ideal solution quickly. Here I simply list several applications.

1. Information exchange

   Let us see an algorithm for information exchange. Suppose a node has some data which it wants to pass on to all the other nodes in the network. It does this by passing a copy of the data to one of its neighbour chosen at random, and repeats this until the data reaches all nodes. In the other words, it starts a random walk to pass that data in the network until the data is received by all nodes. Stephen Boyd [Boy03] did research showing that the time this process takes clearly lower bounds the mixing time of a random walk on that network. So we can use the mixing time to estimate the total time that for the information exchange cost in a certain network. And can be improved by changing the topology structures of the network to reduce the mixing time.

2. Page ranking

   The system for ranking web pages of famous search engine Google uses an ingenious algorithm. They describe all the web pages by a directed graphs. Pages are the nodes and the edges exist if there are links between them. PageRank is determined by π which is the stationary distribution

of the Markov chain on the graph. So suppose π_i is the distribution state at step i, it can be calculated as

$$\pi_{i+1} = dP\pi_i + (1-d)$$

here d is some constant in the range $0 < d < 1$, P is the transition probability matrix of the Markov chain. Desmond and Alan described it in more detail in their paper [HT03]. For a search engine speed is very important. There are thousands of people submitting their search requirements to the server every second, the PageRank process must be fast and accurate. The mixing time tells people how long it will take to ranking these pages, that is, how long would people wait before they get their search results. It's a simple way to evaluate this algorithm.

3. Sampling problem [Gur]

   For many fundamental sampling problems, the most useful and normal way to solve them is to use random walk. Suppose Ω and let π be a probability distribution on Ω. The general 'sampling' problems them to pick an element of Ω randomly according to the distribution π. So to sample according to a distribution π, we do as follows: define a Markov chain which has a unique stationary distribution π. Starting from an arbitrary state in Ω, take a random walk on the Markov chain for some steps, and then output the final state, which is the sample we want. The stationary distribution implies that, by taking the walk long enough, we can ensure that the output state is arbitrarily close to the desired distribution π. So the mixing time is one of the techniques to decide how long the random walk we take will be sufficient to get a sample result which is good enough. So if we can obtain good upper bounds on the mixing time of the chain, we can know at least how many steps we need to get a good sample result.

5.2 Conclusion and future work

This report generally introduces the random walk on the undirected graphs, together with some important properties and focus on the mixing time to measure the convergence speed of a random walk. The mixing time is determined by the second largest eigenvalue modular of adjacency matrix (SELM). Since for large non-symmetric matrix it is not easy to find the eigensystems, I studied the eigensystems of several related matrices, find that the matrix N, which is symmetric and defined by L. Lovász, can give the answer more practical and

easier than any other matrices. This made it practical to calculate the mixing time for large network.

We raised the fastest mixing Markov chain on topology problem based on the practical program to find the SELM of the matrix N for all different topology network (still connected and fixed with the number of nodes). And then I found that Stephen Boyd's fastest mixing Markov chain problem based on changing the weights. Both these two FMMC problems constrain on either fixed weights or fixed topology, so I defined the new fastest reversible (symmetric) mixing Markov chain problem (FRMMC), that is the fastest reversible chain both on topology and weights. It is better since it does not have any constraints on either topology or weights.

I did more research on the effects to mixing time which caused by changing the topology, based on the small world models defined by M. J. Newman. The numerical results got from experiments on model $SW(n, m)$ and $SW\{n, p\}$ shown that the mixing time τ is to the right proportion of n^p and m^q. And the analysis confirmed it. So the convergence speed of a random walk (or Markov chain) is to the proportion of p power to the number of the total nodes and q power to the number of the cross links in the networks, where p and q are some fixed constant.

This is the work I did so far. It will be better if we can get the analytical results of the relations between mixing time and topology. Also more models will tell more information. Although there is not enough time to do all these analyses, it still worthwhile for someone to do more research in this field. Here I will raise some suggestions on future work which might be useful.

1. Analytical proof of the relations between τ and n, m

   I have only derived the relations between τ and n, m numerically. The eigenvalue perturbation might work to find the relations analytically, but there is a problem with repeated eigenvalues, which means eigenvectors are not uniquely determined. The perturbation theory needs the eigenvectors, so we cannot use it unless we know what direction we should take in the eigenspace. So if there is some way to deal with the degenerate eigenvalues, we may get the numerical results proven analytically and get an exact expression for the constants p and q.

2. More models

   Small world is only one of the important network models in real world. There are many other models which are also have very good properties and widely used. Such as scale-free graphs, mesh (torus) graphs and so on. If the mixing time can be studied on these important models, it will be more useful and more applied.

3. Hitting time and mixing time

   I did not do much work on hitting time in this project. There are already many research carried on this field. Hitting time is also an important property of random walk, it related to the spread problem and the transmitting speed. The average hitting time on the whole network somehow explains the connectivity of a network, and should have some strong relations with mixing time.

4. Mixing time and graph statistics

   The graph statistics, like mean shortest path, cluster coefficient, etc., describe the topology structure of a network. We have seen that the topology effects the mixing time in a simple way, so the graph statistics might explain the way that the mixing time changes.

Appendix A

Complete computational results

A.1 Mixing time on small regular graphs

Table A.1: Mixing time on small regular graphs

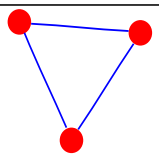
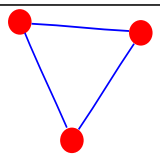
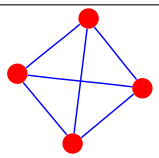
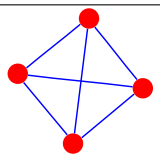
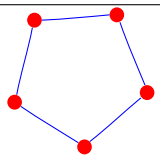
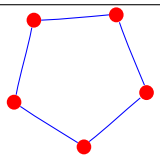
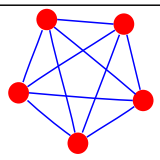
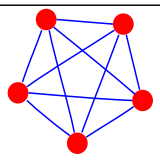
nodes	degree	num	max time	graph	min time	graph	avertime
3	2	1	1.4427		1.4427		1.4427
4	3	1	0.9102		0.9102		0.9102
5	2	1	4.7184		4.7184		4.7184
5	4	1	0.7213		0.7213		0.7213

Table A.1: Mixing time on small regular graphs

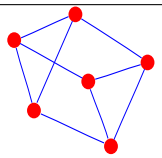
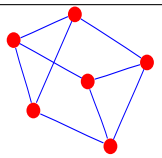
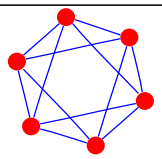
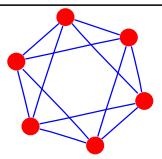
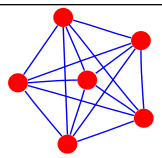
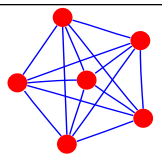
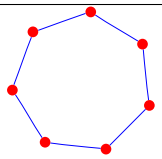
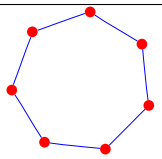
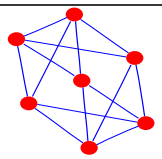
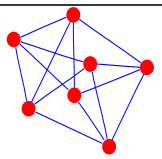
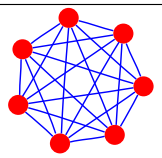
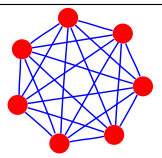
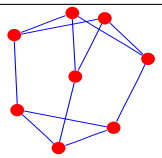
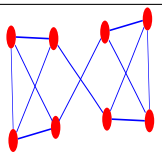
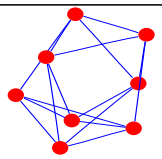
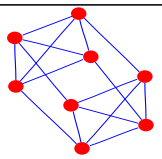
nodes	degree	num	max time	graph	min time	graph	avertime
6	3	1	2.4663		2.4663		2.4663
6	4	1	1.4427		1.4427		1.4427
6	5	1	0.6213		0.6213		0.6213
7	2	1	9.5891		9.5891		9.5891
7	4	2	3.4761		1.7340		2.6050
7	6	1	0.5581		0.5581		0.5581
8	3	4	6.3292		3.4026		4.7346
8	4	5	4.7184		1.4427		2.4941

Table A.1: Mixing time on small regular graphs

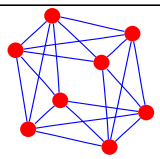
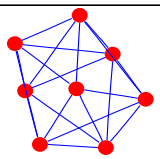
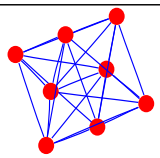
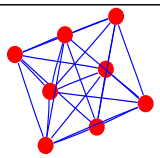
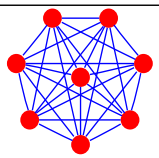
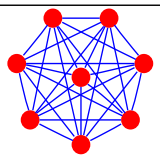
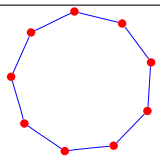
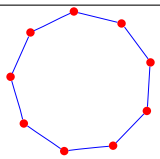
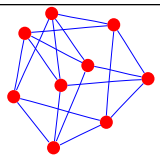
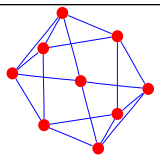
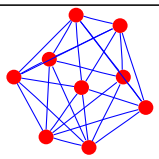
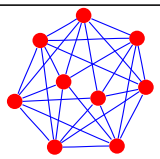
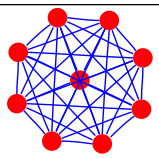
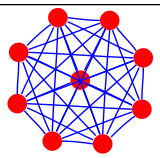
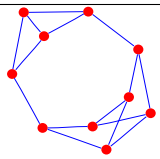
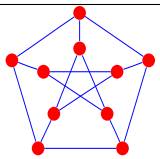
nodes	degree	num	max time	graph	min time	graph	avertime
8	5	3	1.9576		1.3735		1.7629
8	6	1	0.9102		0.9102		0.9102
8	7	1	0.5139		0.5139		0.5139
9	2	1	16.0765		16.0765		16.0765
9	4	16	8.6134		1.4427		3.0804
9	6	4	1.4427		1.1591		1.3718
9	8	1	0.4809		0.4809		0.4809
10	3	17	15.5896		2.4663		6.8216

Table A.1: Mixing time on small regular graphs

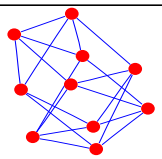
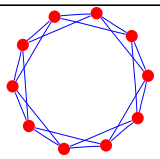
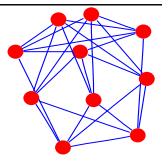
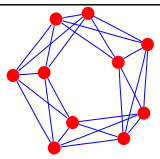
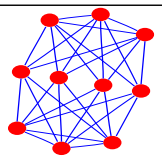
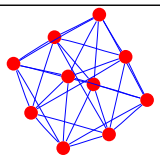
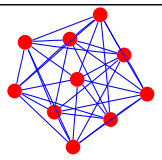
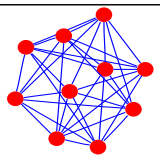
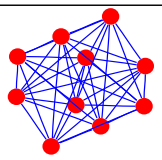
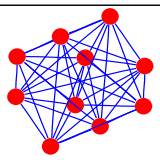
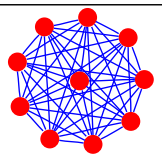
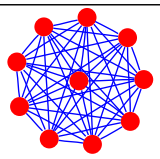
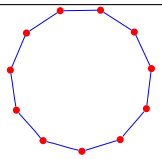
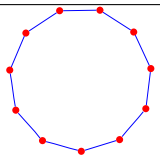
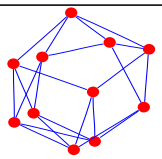
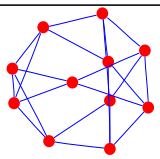
nodes	degree	num	max time	graph	min time	graph	avertime
10	4	58	7.7220		1.7195		3.4093
10	5	59	7.4542		1.2427		2.2145
10	6	21	2.4663		0.9102		1.5722
10	7	5	1.1802		1.0168		1.1475
10	8	1	0.7213		0.7213		0.7213
10	9	1	0.4551		0.4551		0.4551
11	2	1	24.1836		24.1836		24.1836
11	4	265	12.3769		1.7195		3.6497

Table A.1: Mixing time on small regular graphs

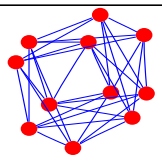
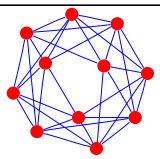
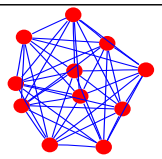
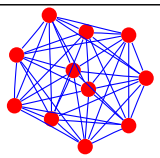
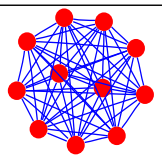
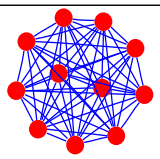
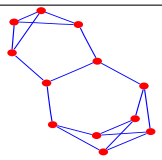
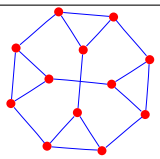
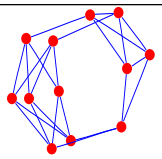
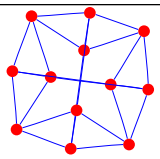
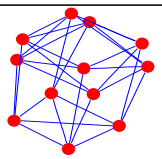
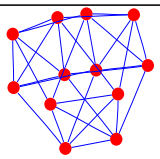
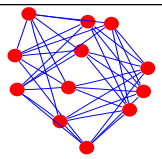
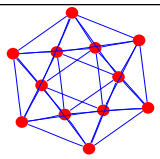
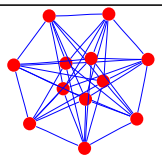
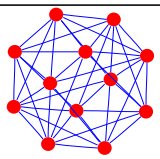
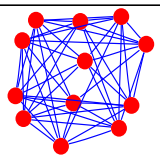
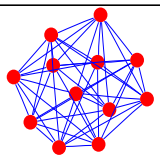
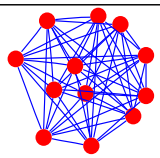
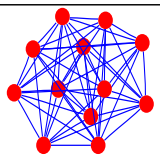
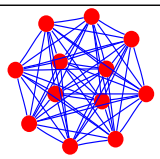
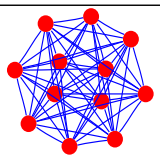
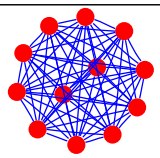
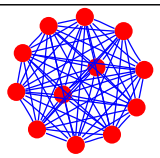
nodes	degree	num	max time	graph	min time	graph	avertime
11	6	266	5.4848		1.0903		1.6889
11	8	6	1.0195		0.9152		1.0022
11	10	1	0.4343		0.4343		0.4343
12	3	80	27.0755		2.4663		8.5713
12	4	1540	14.5177		1.4427		3.7494
12	5	7847	10.8962		1.0914		2.3717
12	6	7848	10.6884		0.9102		1.7613
12	7	1547	2.9720		0.7982		1.4004

Table A.1: Mixing time on small regular graphs

nodes	degree	num	max time	graph	min time	graph	avertime
12	8	94	1.4427		0.8693		1.1517
12	9	9	0.9102		0.8388		0.9023
12	10	1	0.6213		0.6213		0.6213
12	11	1	0.4170		0.4170		0.4170

A.2 FMRMC on small regular graphs

Table A.2: FMRMC on small regular graphs

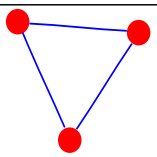
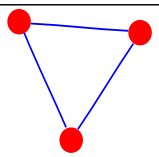
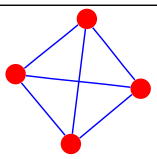
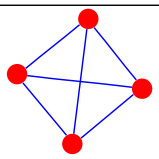
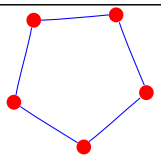
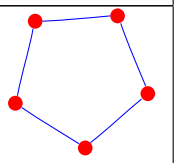
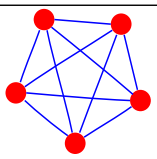
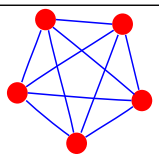
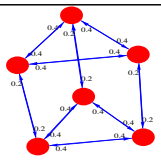
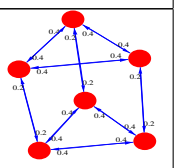
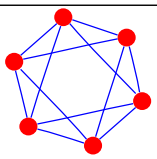
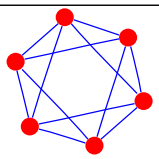
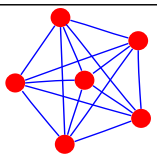
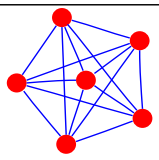
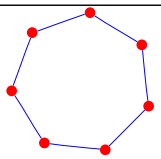
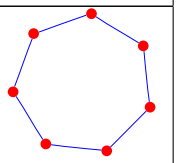
nodes	degree	max time	graph	min time	graph
3	2	1.4427		1.4427	
4	3	0.9102		0.9102	
5	2	4.7184		4.7184	
5	4	0.7213		0.7213	
6	3	1.9576		1.9576	
6	4	1.4427		1.4427	
6	5	0.6213		0.6213	
7	2	9.5891		9.5891	

Table A.2: FMRMC on small regular graphs

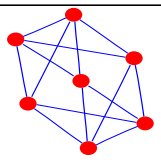
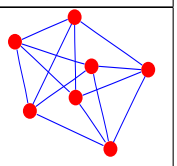
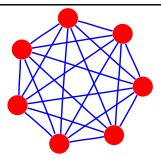
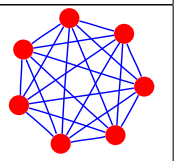
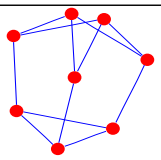
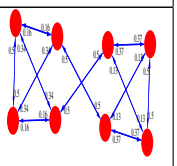
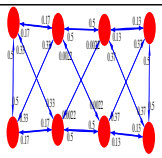
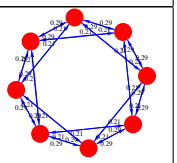
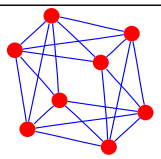
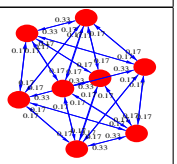
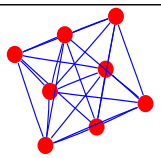
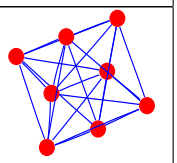
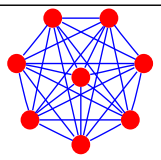
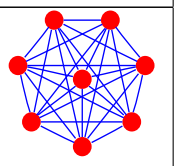
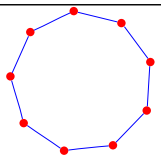
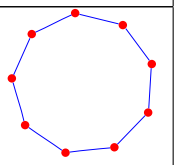
nodes	degree	max time	graph	min time	graph
7	4	3.4761		1.1897	
7	6	0.5581		0.5581	
8	3	6.3292		2.8854	
8	4	2.8854		1.1344	
8	5	1.9576		0.9102	
8	6	0.9102		0.9102	
8	7	0.5139		0.5139	
9	2	16.0765		16.0765	

Table A.2: FMRMC on small regular graphs

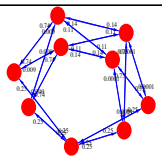
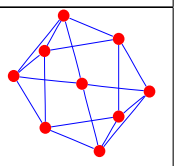
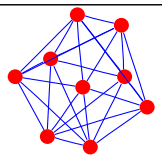
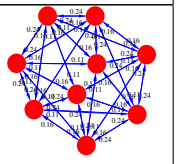
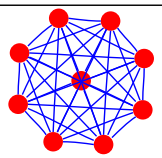
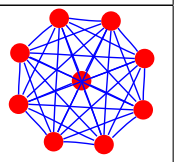
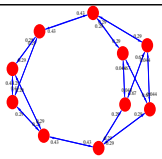
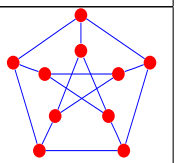
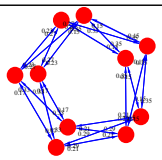
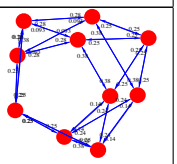
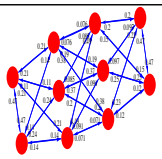
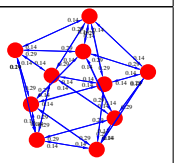
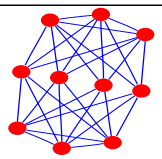
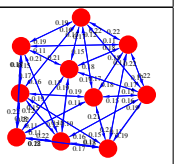
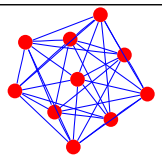
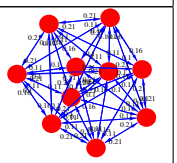
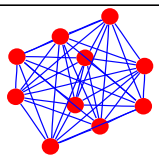
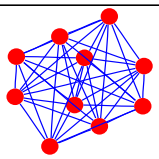
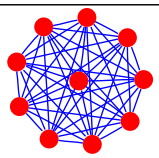
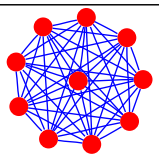
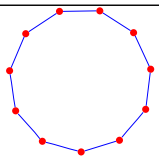
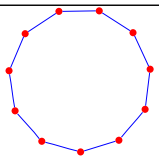
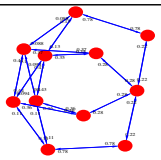
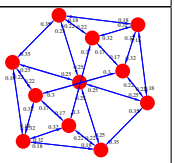
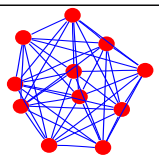
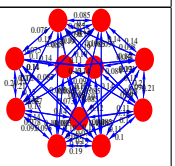
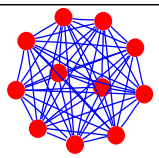
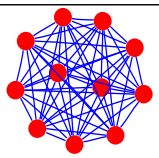
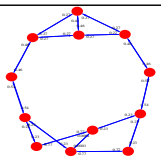
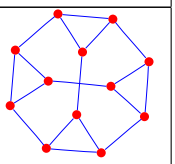
nodes	degree	max time	graph	min time	graph
9	4	8.2117		1.4427	
9	6	1.4427		0.8662	
9	8	0.4809		0.4809	
10	3	14.9392		2.4663	
10	4	4.7185		1.5767	
10	5	4.3522		1.1802	
10	6	2.4663		0.9064	
10	7	1.1802		0.7670	

Table A.2: FMRMC on small regular graphs

nodes	degree	max time	graph	min time	graph
10	8	0.7213		0.7213	
10	9	0.4551		0.4551	
11	2	24.1836		24.1836	
11	4	10.9225		1.5167	
11	8	1.0195		0.6953	
11	10	0.4343		0.4343	
12	3	23.3497		2.4663	

Appendix B

Computional method

B.0.1 pysparse

`Pysparse` is the package I used in my programs to calculate the eigensystems for large matrices. Since the networks I did in the simulation are very large, it is not easy to solve the eigensystems for these huge matrices very efficiently. `Pysparse` is a `Python` software package for manipulating sparse matrices. It can store symmetric matrices efficiently and includes the modules that implement a Jacobi-Davidson eigenvalue solver for the symmetric, generalised matrix eigenvalue problem (JDSYM).

The code example below illustrates the use of the new sparse matrix and the way to use JSDYM.

```
import spmatrix,itsolvers,precon,jdsym

A=spmatrix.ll_mat_sym(5,5)
for i in range(5):
  A[i,i] = i
A[3,0]= A[3,3]
tau=1.0

sol=jdsym.jdsym(A,M,K,5,tau,jdtol=1e-9,itmax=500,linsolver=itsolvers.qmrs)
```

The `jdsym` module function will return a tuple with four elements. The second one are the eigenvalues and the third one is the eigenvectors. More information and further code examples for Pysparse can be found in [Geu02] and [GA03]

B.0.2 geng

`Geng` is a program to generate small graphs from the nauty package [McK]. Under given constraints `geng` can generate all the different graphs (not including isomorphic ones) satisfies the conditions. These graphs are in the format of graph6. graph6 is format for storing undirected graphs in a compact manner, which is suitable for small graphs, using only printable ASCII characters. Files in these formats have text type and contain one line per graph. There is a stand-alone reader `showg` program which can convert them into adjacency list, which can be easily convert to many other formats.

For example, if we want to generate all 10 nodes regular degree 3 connected graphs, the command is as follows:

```
geng −c 10 −d3 −D3 | showg
```

Type the command in the console in Linux platform; the program will return the adjacency lists for all the connected regular 3 graphs of 10 nodes without isomorphic ones.

B.0.3 sdpsol

The program `sdpsol` parses semidefinite programming (SDP) and determinant maximization (MAXDET) problems expressed in the `sdpsol` language, solve them using interior-point algorithms, and reports in a convenient form. The MAXDET problem (including its special case, SDP) is a convex optimization problem, i.e., the objective function is convex and the constraint set is convex. I use this package in the program to find the fastest reversible Markov chain in our combined fastest mixing problem. More detail can be found in [WB96].

Appendix C

Program listings

C.0.4 Calculate the hitting time matrix

```
#!/usr/local/bin/python
# cat foo.dat | h_time17.py
# this version is also suitable for multigraphs.
# Read the adjacency dictionary from stdin, print the hitting time matrix

import spmatrix,itsolvers,precon,jdsym,Numeric
import numarray as na
from math import sqrt,log,exp
from array import array
from sys import stdin,stderr,argv

def sqrtD(A):
  degree,sqrtD=na.resize(na.array([0.0]),(n)),na.resize(na.array([0.0]),(n))
  for i in range(n):
    for j in range(n):
      if A[i,j]!=0:
        degree[i]+=A[i,j]
    if degree[i]!=0:
      sqrtD[i]=sqrt(1.0/degree[i])
  return (degree,sqrtD)

def adjd2adjm(adj):
  A=spmatrix.ll_mat_sym(n,n)
  for node in adj:
    for neighbour in adj[node]:
      if node>=neighbour:
        A[node,neighbour]+=1
```

```
  return(A)

def h_time(A):
  M,K=None,None
  N=spmatrix.ll_mat_sym(n,n)
  for j in range(n):
    for i in range(j,n):
      if sq[i] and sq[j] and A[i,j]:
        N[i,j]=sq[i]*A[i,j]*sq[j]
      else:
        N[i,j]=0
  sol=jdsym.jdsym(N,M,K,n,1,jdtol=1e-9,itmax=500,linsolver=itsolvers.qmrs)
  p=heapsort(sol[1])
  lmbd=sol[1].tolist()
  if abs(lmbd[1]-1)<1e-6:
    sol=jdsym.jdsym(N,M,K,n,0.999999,jdtol=1e-9,itmax=500,linsolver=itsolvers.qmrs)
    p=heapsort(sol[1])
    lmbd=sol[1].tolist()
  if abs(lmbd[1]-1)>1e-6:
    v=na.transpose(sol[2].tolist())
    H=spmatrix.ll_mat(n,n,n)
    for k in range(n):
      lmbdk=lmbd[k]
      for i in range(n):
        di=degree[i]
        xki=v[p[k]][i]
        for j in range(n):
          dj=degree[j]
          xkj=v[p[k]][j]
          if k!=0:
            H[j,i]+=m*(xki**2/di-xki*xkj/sqrt(di*dj))/(1-lmbdk)
    return H

def downheap(a,p,v,n):    # v is the start vertex
  ' helper function for heapsort '
  w=2*v+1                 # first descendant of v
  while w<n:
    if w+1<n:             # is there a second descendant?
      if a[w+1]<a[w]: w+=1 # sort down
    # w is the descendant of v with maximum label
    if a[v]<=a[w]: return # v has the heap property
    # otherwise
    a[w],a[v]=a[v],a[w]
    p[w],p[v]=p[v],p[w]
    v=w; w=2*v+1          # continue

def heapsort(a):   # sort a
```

```
  n=len(a)
  p=[i for i in range(n)]
  for v in range(n/2-1,-1,-1):
    downheap(a,p,v,n)
  for v in range(n-1,0,-1):
    a[0],a[v]=a[v],a[0]
    p[0],p[v]=p[v],p[0]
    downheap(a,p,0,v)
  return p # permutation

def printmatrix(a):
  for i in range(n):
    for j in range(n):
      print '%8.3f'%a[i,j],'',
    print

def printvector(a):
  n=len(a)
  for i in range(n):
    print '%8.3f'%a[i],'',
  print


count=0
for line in stdin:
  count+=1
  adj=eval(line)
  n=len(adj)
  A=adjd2adjm(adj)
  degree,sq=sqrtD(A)
  m=sum(degree)
  H=h_time(A)
  if H!=None:
    print 'adj%d=%s'%(count,adj)
    print 'Hitting time matrix for graph%d:'%count
    printmatrix(H)
    check=0
    for node in adj:
      for neighbour in adj[node]:
        check+=H[node,neighbour]
    print '%8.3f=?=%d'%(check,(n-1)*m)
  else:
    print 'pysparse erro with 2nd lambda=1'
  print
```

C.0.5 Calculate the mixing time of the network

```
#!/usr/local/bin/python
# This program reads the adjacency dictionary of the graph from stdin, then
# calculate the mixing rate and mixing time.
# geng -c 6 6:6 | showg -e | ./showg-e2adj1.py 2>/dev/null | ./mixingtime.py
# generate graphs | convert to showg format | generate the adjacency dictionary
# | read the adjacency dictionary to calculate the mixing time.

import spmatrix,itsolvers,precon,jdsym
from sys import stdin,stderr,argv
from math import sqrt,log
from commands import getstatusoutput
from numarray import array,resize

def adjd2adjm(adj): # convert the adjacency dictionary to adjacency matrix
  A=spmatrix.ll_mat_sym(n,n)
  N=spmatrix.ll_mat_sym(n,n)
  degree=resize(array([0]),(n))
  sqrtD=resize(array([0.0]),(n))
  for node in adj:
    degree[node]=len(adj[node])
    sqrtD[node]=sqrt(1.0/degree[node])
    for neighbour in adj[node]:
      if node>=neighbour:
        A[node,neighbour]+=1
  for j in range(n): # generate the matrix N
    for i in range(j,n):
      if sqrtD[i] and sqrtD[j] and A[i,j]:
        N[i,j]=sqrtD[i]*A[i,j]*sqrtD[j]
    N[j,j]=N[j,j]
  return (N,degree)

def mixing_rate(N):   # find the mixing rate
  M,K=None,None
  tau1=1.0
  tau2=-1.0
  sol1=jdsym.jdsym(N,M,K,2,tau1,jdtol=1e-9,itmax=500,linsolver=itsolvers.qmrs)
  sol2=jdsym.jdsym(N,M,K,1,tau2,jdtol=1e-9,itmax=500,linsolver=itsolvers.qmrs)
  # find the second largest absolute of eigenvalues
  mu=abs(max(abs(sol1[1][1]),abs(sol2[1][0])))
  if mu<1-(1e-6) and mu>1e-6:
    m_rate=log(1/mu)
    return m_rate
  else: return 0
```

```
for line in stdin:
  adj=eval(line)
  n=len(adj)
  N,degree=adjd2adjm(adj)
  mrate=mixing_rate(N)
  if mrate:
    print 'mixing rate',mrate
    print 'mixing time ',1/mrate
  else: print 'SLEM is out of range'
```

C.0.6 Fastest reversible mixing Markov chain

```
# KMB 2004 Jul 15
# fastest mixing reversible Markov chain
#   geng 4 -c | showg -e -l0 | fmrmc.py -v2
#   geng 7 -c | showg -e -l0 | fmrmc.py -v1 | kw mixtime | histogram | p
#   geng 8 -c -d3 -D3 | showg -e -l0 | time fmrmc.py -v1 | kw mixtime | histogram | p
#   Petersen: geng    -c 10 -d3 -D3 | showg -e -l0 | time fmrmc.py
#   line graph: geng 6 -c 5:5 -d1 -D2      | showg -e -l0 | fmrmc.py
#   geng 10 -c -d3 -D3| showg -e -l0 | showg-e2dot.py > n10d3.dot
#   neato -Tps n10d3.dot > n10d3.ps

import re
import getopt
from sys import stdin,stderr,exit,argv
from copy import copy
from commands import getstatusoutput
from math import sqrt,log

# 0=>no output
# 1=>mixtime only
# 2=>even more
# 3=>everything
verbosity=0
sl=False # self-loops not allowed
maxng=10000000

try:
  opts,args = getopt.getopt(argv[1:],'sv:n:',['selfloops','verbosity','maxng'])
except getopt.GetoptError:
  print >>stderr,'Bad options'
  exit(2)
for o,a in opts:
  if o in ("-v", "--verbosity"):
```

```
    verbosity=int(a)
  if o in ("-s", "--selfloops"):
    sl=True # self-loops allowed
  if o in ("-n", "--maxng"):
    maxng=int(a)
    print >>stderr,'maxng=%d'%maxng

if sl: print >>stderr,'self-loops allowed'                                    40
else:  print >>stderr,'self-loops not allowed'

def writesdpprog(f,adj,aa,deg,sl=True,nu=10):
  ' aa=antiadjacency dict, sl=>self loops allowed '
  n=len(aa) # nodes
  m=sum([n-len(aa[i]) for i in range(n)])/2     # links
  f.write('''%% fastest mixing reversible Markov chain
  %% automatically generated by fmrmc.py
  %% deg=%s
  %% adj=%s (indexed from 0)                                                  50
  %%MAXITER=100;
  %%ABSTOL=1e-8;
  %%RELTOL=1e-6;
  %%BIGM=1000;
  NU=%d;
  n=%d;
  variable s,P(n,n); %% these are indexed from 1
  %%initialize s=5;
  P.>-1e-14;\n'''%(str(deg),str(adj),nu,n))
  # constraints...                                                           60
  for x in aa.keys():
    for y in aa[x]:
      if x!=y:
        f.write('   P(%d,%d)==0;\n'%(x+1,y+1))
      elif not sl: # x==y, and self-loops not allowed
        f.write('   P(%d,%d)==0;\n'%(x+1,y+1))
  # initialization...
  P=' initialize P=['
  pi=' pi=['
  q=' q=['                                                                     70
  Pih=' Pih=['
  Pimh=' Pimh=['
  for x in range(n):
    degx=deg[x]
    pi+='%.14g,'%(0.5*degx/m)
    q+='%.14g,'%sqrt(0.5*degx/m)
    for y in range(n):
      if x==y:
        Pih +='%.14g,'%(sqrt(0.5*degx/m))
```

```
        Pimh+='%.14g,'%(1.0/sqrt(0.5*degx/m))
      else:
        Pih+='0,'
        Pimh+='0,'
      if y in aa[x]: # not y~x
        P+='0,'
      else: # y~x
        P+='%.14g,'%(1.0/degx)
    P=P[:-1]+'; '
    pi=pi[:-1]+'; '
    q=q[:-1]+'; '
    Pih=Pih[:-1]+'; '
    Pimh=Pimh[:-1]+'; '
  P=P[:-2]+'];\n'
  pi=pi[:-2]+']; %% pi=desired invariant distribution, Pi=diag(pi)\n'
  q=q[:-2]+'];\n'
  Pih=Pih[:-2]+']; %% Pi^(1/2)\n'
  Pimh=Pimh[:-2]+']; %% Pi^(-1/2)\n'
  f.write(P)
  f.write(pi)
  f.write(Pih)
  f.write(Pimh)
  f.write(q)
  f.write('''
  P*ones(n)==ones(n);
  diag(pi)*P==P'*diag(pi);
  B=Pih*P*Pimh-q*q';
  B< s;
  B>-s;
  minimize xxxx=s;
  ''')

def a2aa(a):
  n=len(adj)
  x=[i for i in range(n)]
  aa=dict([(i,copy(x)) for i in range(n)])
  for x in a:
    for y in a[x]: aa[x].remove(y)
  return aa

graph=re.compile(r'^Graph (\d+),')
nm=re.compile(r'(\d+)\s+(\d+)')
ed=re.compile(r'\s*(?P<a>\d+)\s+(?P<b>\d+)')
ob=re.compile(r'xxxx\s+=\s+(?P<ob>.*)')
pm=re.compile(r'^(?P<pm>\[.*?\])',re.DOTALL|re.MULTILINE)
mintau= 1e300
maxtau=-1e300
```

```
ng=0
ok=[]

while ng<maxng:                                                    130
  line=stdin.readline()
  if not line: break
  x=graph.match(line)
  if x:
    ng+=1
    line=stdin.readline()
    y=nm.match(line)
    if not y: exit(1)
    if verbosity and ng%1000==0: print >>stderr,ng
    ve=tuple(map(int,y.groups())) # new graph, # vertices and # edges  140
    if verbosity>1: print 'graph',x.groups(0)[0],' n=%d m=%d'%ve
    adj=dict([(i,[]) for i in range(ve[0])])
    n=len(adj)
    line=stdin.readline()
    ne=0
    while ne<ve[1]:
      for e in re.finditer(ed,line):
        ne+=1
        a,b=int(e.group('a')),int(e.group('b'))
        adj[a].append(b)                                           150
        adj[b].append(a)
    aa=a2aa(adj)
    if verbosity>1: print '  adjacency dictionary: ',adj
    if verbosity>2: print '  antiadjacency dictionary: ',aa
    deg=[len(adj[i]) for i in range(n)]
    Dmh=[sqrt(deg[i]) for i in range(n)]
    Dh =[sqrt(1.0/deg[i]) for i in range(n)]
    if verbosity>2: print 'deg=',deg,' D^(-1/2)=',Dmh
    aa=a2aa(adj)
    f=file('foo.sdp','w')                                          160
    if    n>=9: writesdpprog(f,adj,aa,deg,sl,3)
    elif n>=7: writesdpprog(f,adj,aa,deg,sl,8)
    elif n==6: writesdpprog(f,adj,aa,deg,sl,40)
    elif n==4:
      if ng==3: writesdpprog(f,adj,aa,deg,sl,3)
      else:       writesdpprog(f,adj,aa,deg,sl,3)
    else: writesdpprog(f,adj,aa,deg,sl,4)
    f.close()
    s,o=getstatusoutput('sdpsol foo.sdp')
    s=s>>8                                                         170
    if not s in [55,62] and verbosity>0:
      print >>stderr,' sdpsol exit status=',s
    if verbosity>2: print >>stderr,'  sdpsol output P=',o
```

```
    m=ob.search(o)
    if s in [55,62] and m:
    # 55 seems to mean ok, 62=>FEASIBLE but might be NON-OPTIMAL after 100 iterations,
    # sdpsol stopped because MAXIMUM NUMBER OF ITERATIONS exceeded.
      mu=float(m.group('ob'))
      if mu<1.0:
        ok.append(ng)                                                          180
        tau=-1.0/log(mu)
        if verbosity>1: print '   mu=%g'%mu
        if verbosity>0: print '   graph %d: mixtime=%g'%(ng,tau)
        if tau<mintau:
          mintau=tau
          mintaug=ng
          mintauadj=copy(adj)
          p=pm.search(o)
          if p: mintauP=p.group('pm')
          else: mintauP=o                                                      190
        if tau>maxtau:
          maxtau=tau
          maxtaug=ng
          maxtauadj=copy(adj)
          p=pm.search(o)
          if p: maxtauP=p.group('pm')
          else: maxtauP=o
      else:
        print >>stderr,'graph %d: mu=%g >= 1'%(ng,mu)
    elif verbosity>1 and not s in [55,62]: # some problem with sdpsol          200
      if s in [255]:
        print 'sdpsol problem: (fatal) equality constraints given are inconsistent.'
      else:
        print 'sdpsol problem:'
        print o

print >>stderr,'%d graphs processed, nok=%d'%(ng,len(ok))
if verbosity>0:
  print >>stderr,'ok=',ok
                                                                             210
def stats(f):
  if mintau<1e100:
    f.write('\nminimum fastest mixing time=%g\n'%mintau)
    f.write('minimum fastest mixing graph number=%d\n'%mintaug)
    f.write('minimum fastest mixing graph = %s\n'%str(mintauadj))
    f.write('minimum fastest mixing sdpsol output P=\n')
    f.write(mintauP)
    f.write('\n')
  if maxtau>-1e100:
    f.write('\nmaximum fastest mixing time=%g\n'%maxtau)                      220
```

```
    f.write('maximum fastest mixing graph number=%d\n'%maxtaug)
    f.write('maximum fastest mixing graph = %s\n'%str(maxtauadj))
    f.write('maximum fastest mixing sdpsol output P=\n')
    f.write(maxtauP)
    f.write('\n')

stats(stderr)
if sl: print >>stderr,'self-loops were allowed'
else:  print >>stderr,'self-loops were not allowed'
```

C.0.7 Generate the graphs from the $SW(n, m)$ model

```
from RandomArray import poisson

def adjd4swnm(n,m):
  adj=dict([(i,[(i+n-1)%n,(i+1)%n]) for i in range(n)])
  ncl=0
  while ncl<m:
    a,b=randint(0,n-1),randint(0,n-1)
    if a!=b and a!=(b+n-1)%n and a!=(b+1)%n:
      adj[a].append(b)
      adj[b].append(a)
      ncl+=1
  return adj
```

C.0.8 Generate the graphs from the $SW\{n, p\}$ model

```
from random import randint

def adjd4swnp(n,p):
  meanm=n*(n-1)*p/2
  mvec=poisson(meanm,[200])
  adj=[]
  for m in mvec:
    adj.append(adjd4swnm(n,m))
  return adj
```

Bibliography

[AB01] Réka Albert and Albert-László Barabási. Statistical mechainics of complex networks. 2001. http://arxiv.org/abs/cond-mat/0106096.

[Bar01] Albert-László Barabási. The physics of the web. *Physics World*, 2001. http://physicsweb.org/article/world/14/7/9/1/pw1407091.

[BDX03] Stephen Boyd, Persi Diaconis, and Lin Xiao. Fastest mixing Markov chain on a graph. *SIAM Review*, 2003. http://www.stanford.edu/~boyd/reports/fmmc.pdf.

[Big93] Norman Biggs. *Algebraic graph theory*. Cambridge Mathematical Library, 2nd edition, 1993.

[Bol98] Béla Bollobás. *Modern Graph Theory*. Springer, 2nd edition, 1998.

[Bol01] Béla Bollobás. *Random Graphs*. CUP, 2nd edition, 2001.

[Boy03] Stephen Boyd. Gossip and mixing times of random walks on random graphs. 2003. http://www.stanford.edu/~boyd/reports/gossip_gnr.pdf.

[BRST01] Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. The degree sequence of a scale-free random graph process. *Random Struct. Alg.*, 18:279–290, 2001. http://www.math.cmu.edu/~af1p/WWW2004/swdeg.pdf.

[Chu97] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1st edition, 1997.

[Die00] Reinhard Diestel. *Graph Theory*. Springer, 2nd edition, 2000.

[GA03] Roman Geus and Peter Arbenz. Pysparse and pyfemax: A python framework for large scale sparse linear algebra. 2003. http://www.inf.ethz.ch/~arbenz/pycon03_contrib.pdf.

[Geu02] Roman Geus. Pysparse - a sparse linear algebra extension for python. 2002. http://pysparse.sf.net.

[GN04] Michael T. Gastner and M. E. J. Newman. The spatial structure of networks. 2004. http://arxiv.org/abs/cond-mat/0407680.

[GS92] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes*. Oxford Science, 2nd edition, 1992.

[Gur] Venkatesan Guruswami. Rapidly mixing Markov chain: a comparison of techniques. http://www.cs.washington.edu/homes/venkat/pubs/papers/markov-survey.ps.

[Hig01] Desmond J. Higham. A small world phenomenon for Markov chains. 2001. http://www.maths.strath.ac.uk/~aas96106/rep22_2001.ps.

[HJ90] R. A. Horn and C. R. Johnson. *Matrix Analysis*. CUP, 1990.

[HT03] Desmond J. Higham and Alan Taylor. The sleekest link algorithm. *(IMA) Mathematics Today*, 2003. http://www.ece.northwestern.edu/~nocedal/328/report.pdf.

[Lov93] L. Lovász. Random walks on graphs: a survey. *Combinatorics*, 2:353–398, 1993. http://research.microsoft.com/users/lovasz/erdos.ps.

[McK] Brendan D. McKay. nauty. http://cs.anu.edu.au/~bdm/nauty/.

[MK04] Naoki Masuda and Norio Konno. Return times of random walk on generalized random graphs. *Physics Review*, 69, 2004.

[Mon99] Rémi Monasson. Diffusion, localization and dispersion relations on 'small-world' lattices. *Eur. Phys.*, 12:555–567, 1999. http://www.edpsciences.org/articles/epjb/abs/1999/24/b9213/b9213.html.

[New02] M. E. J. Newman. Random graphs as models of networks. 2002. http://arxiv.org/abs/cond-mat/0202208.

[New03] M. E. J. Newman. The structure and function of complex networks. 2003. http://arxiv.org/abs/cond-mat/0303516.

[Nor02] Stephen C. North. Drawing graphs with neato. 2002. http://www.research.att.com/sw/tools/graphviz/neatoguide.pdf.

[PA00] Sagar A. Pandit and R. E. Amritkar. Random spread on the family of small-world networks. 2000. http://arxiv.org/abs/cond-mat/0004163.

[WB96] Shao-Po Wu and Stephen Boyd. sdpsol a parser/solver for semidefinite programming and determinant maximization problems with matrix structure. 1996. http://www.stanford.edu/~boyd/sdpsol/usrguide.pdf.

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998. http://tam.cornell.edu/SS_nature_smallworld.pdf.